2004

Multiprocessor DSP Implementation of the JPEG 2000 Codec

Scott Mayne

Multiprocessor DSP Implementation of the JPEG2000 Codec

by

Scott M. Mayne

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Associate Professor Dr. Muhammad Shaaban
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
November 2004

Approved By:

Dr. Muhammad Shaaban
Associate Professor, Department of Computer Engineering
Rochester Institute of Technology
Primary Adviser

Dr. Juan Cockburn
Associate Professor, Department of Computer Engineering
Rochester Institute of Technology

Norm Zeck
Technical Lead, Imaging Systems Area
Wilson Center for Research and Technology
Xerox Corporation

Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

Title: Multiprocessor DSP Implementation of the JPEG2000 Codec

I, Scott M. Mayne, hereby grant permission to the Wallace Memorial Library reproduce my thesis in whole or part.

Scott M. Mayne

November 11, 2004

Date

Acknowledgments

I would like to thank my advisor Dr. Muhammad Shaaban, and thesis committee members Dr. Juan Cockburn and Norm Zeck, for the guidance and mentoring that has made this work possible.

Thank you to Xerox Corporation for providing the hardware and resources needed for this project. A special thanks goes out to everyone at Xerox and RIT who has provided assistance during the course of this work. From finding a topic to final proofreading and every step in between, your help is greatly appreciated.

Last, but not least, I would like to thank my family for their support and encouragement that has helped me complete this thesis.

Abstract

The transition to JPEG2000 from other image formats such as standard JPEG offers improved compression and image quality, yet has not been widely adopted in practice. This is mainly due to the complexity of the JPEG2000 algorithm. Standard JPEG uses the Discrete Cosine Transform (DCT) and Huffmann encoding to achieve its compression, whereas JPEG2000 uses the wavelet transform and arithmetic encoding. Due to the wide acceptance of JPEG, there are processors such as Equator Technology's BSP-15 digital signal processor (DSP) that have been designed with features specifically for JPEG applications. For some of the current digital printing applications where JPEG is used, images must be encoded and decoded at rates exceeding 100 pages per minute. A multiprocessor environment consisting of Equator Technology's BSP-15 processors may offer acceptable performance for the JPEG2000 codec.

The aim of this work is to design a JPEG2000 codec for the BSP-15 processor and to determine if this processor is capable of delivering the performance required by high end digital printers. The features of the BSP-15 that are well suited for the JPEG2000 algorithm will be discussed, as well as future improvements that could be incorporated into the architecture. By analyzing the advantages and disadvantages of this processor, the next generation of processors may be able to offer features that will allow it to excel in JPEG2000 processing.

A multiprocessor DSP implementation of the JPEG2000 codec is the main result of this work. The resulting codec is able to provide more than double the processing throughput of existing JPEG2000 software.

Contents

List of Figures

List of Tables

Glossary

B

BSP Broadband Signal Processor, p. 2.

C

codeblock A rectangular grouping of coefficients from the same subband of a tile component, p. 4.

CREW Compression with Reversible Embedded Wavelets - An algorithm designed by Ricoh Innovations, Inc. for lossless and near-lossless image compression, p. 2.

D

DataStreamer A BSP-15 on-chip DMA controller, p. 25.

DCT Discrete Cosine Transform - used in standard JPEG, p. 1.

discrete dyadic wavelet transform A transformation converting an input signal into two components corresponding to different frequency components, p. 4.

DMA Controller Direct Memory Access Controller - A processor that performs memory-to-memory transfers, p. 25.

DSP Digital Signal Processor, p. 2.

L

lifting steps A technique used to reduce the amount of computation needed to perform the convolution with wavelet kernels, p. 8.

LPS Less Probable Symbol in the binary arithmetic coder, p. 19.

LSig Codeblock coefficient state variable that signifies the coefficient has been included in a previous magnitude refinement coding pass and used to simplify context modeling, p. 16.

M

MPS More Probable Symbol in the binary arithmetic coder, p. 19.

MQ-coder Arithmetic encoding algorithm that is used in JPEG2000, p. 7.

P

packet A part of the bit stream comprising a packet header and the compressed image data from one resolution level of one tile component, p. 4.

Q

Qe Probability of the LPS in the binary arithmetic coder, p. 19.

S

SIMD Single Instruction Multiple Data - Data level parallelism technique, p. 23.

subband A group of transform coefficients resulting from the same sequence of high pass and low pass filtering operations, p. 6.

T

tile A rectangular array of points within an image, p. 4.

V

VLIW Very Long Instruction Word - A type of superscaler computer architecture, p. 23.

VLx Veriable Length encoder/decoder - A coprocessor on the BSP-15 chip, p. 25.

Chapter 1

Introduction

The question

> *Can the advantages of the JPEG2000 standard overcome the algorithm complexity to become widely adopted in high performance digital imaging applications?*

has been asked by imaging experts since the standard was introduced.

Image compression has long been a focus for computer systems. Due to the constraints of storage space and bandwidth restrictions, images must be compressed before saving them to disk or transmitting them over a network. A typical full-page image sent to a printer would require over 128 megabytes of data to be sent from the submitting computer to the printer. Image compression is used to reduce the amount of data needed to represent the image.

Image compression must also take into account the image quality. As compression ratios (the relationship between the original image size and the compressed image size) increase, errors are introduced. For the widely used JPEG format [21], these errors take the form of blocking artifacts and "ringing" near sharp edges. Blocking artifacts are the result of having each 8x8 pixel-block compressed independently. When tight compression constraints are introduced, the continuity between blocks is reduced resulting in a visible edge on block boundaries. Ringing is the result of the periodic nature of the Discrete Cosine Transform (DCT) that is used to reduce the amount of data that needs to be encoded.

An impulse signal in one area of the 8x8 pixel-block affects all other values within the block. Ringing artifacts become particularly visible around synthetic image data such as text. At low bit rates these JPEG artifacts become very objectionable. JPEG2000 introduces a format that produces high quality images even at low bit rates.

JPEG2000 was motivated by a submission by Ricoh Innovations, Inc. to the then developing standard JPEG-LS. Ricoh's Compression with Reversible Embedded Wavelets (CREW) algorithm was not chosen as the basis for JPEG-LS, but sparked interest in a new standardization effort [3]. Twenty-four JPEG2000 candidate algorithms were analyzed at lossless compression and lossy compression at 0.0625, 0.125, 0.25, 0.5, 1, and 2 bits per pixel. The wavelet/trellis coded quantization (WTCQ) algorithm [20] submitted by Science Applications International Corporation (SAIC) and the University of Arizona (UA) was ranked first in both subjective and objective comparisons and chosen as the basis for JPEG2000.

WTCQ was adapted during a series of "Verification Model" (VM) releases that introduced features for error resiliency, decreased memory usage, and flexibility into what is now known as JPEG2000. The JPEG2000 Part 1 (basic features) reached Final Committee Draft status in March 2000 and Final Publication Draft in July 2002.

Experts were predicting early on that the transition from current image formats to JPEG2000 would be rapid. A May 26, 2000 article in the EE Times, speculated that JPEG2000 would be the mainstream format within a couple of years [30]. The complexity of the algorithm has proven difficult to overcome and JPEG2000 has not reached the acceptance that was expected.

This work will focus on designing a JPEG2000 codec implementation for a specialized Equator Technologies BSP-15 DSP platform [12] and determining if this hardware is sufficient to support high performance imaging applications. Performance analysis results will also target areas where there is room for improvement in future models of the DSP. With the support of advanced hardware, JPEG2000 may be able to achieve wider acceptance in the digital imaging industry.

The rest of this document is structured as follows: Chapter 2 will introduce the JPEG2000 algorithm. Chapter 3 will discuss the platform for which the codec was designed. Chapter 4 will discuss the way that the implementation was tailored to map the algorithm to the platform. Chapter 5 will discuss the performance of the codec designed as compared to existing implementations. Chapter 6 will discuss the conclusions drawn from this project and areas for future work.

Chapter 2

The JPEG2000 Algorithm

In this chapter the JPEG2000 algorithm is presented in detail. This will show the areas of the algorithm that account for the large amounts of memory bandwidth and processing that are required to process JPEG2000 images. JPEG2000 Part 1 (referred to henceforth as JPEG2000) describes the compressed codestream as well as other options such as a JP2 file format wrapper. Focus will be given to the required components of the compressed codestream for a single image component (e.g. grayscale images). Be aware that there are features that are not discussed, such as region of interest coding. For the full format specification, refer to [17].

2.1 Introduction

The JPEG2000 algorithm has two major steps that are performed to encode images. The first is an N-level discrete dyadic wavelet transform. The second step arithmetically encodes the bit planes of the transformed data. The data is partitioned at different levels as this processing is performed. These partitions include tiles, packets, and codeblocks. Figure 2.1 illustrates how these elements are combined to form the compressed representation of the image.

The image is first divided into equal sized tiles that will be coded independently. The tiles along the border of the image may be a different size depending on the image dimensions. Each tile undergoes a series of wavelet decomposition steps. The wavelet transform

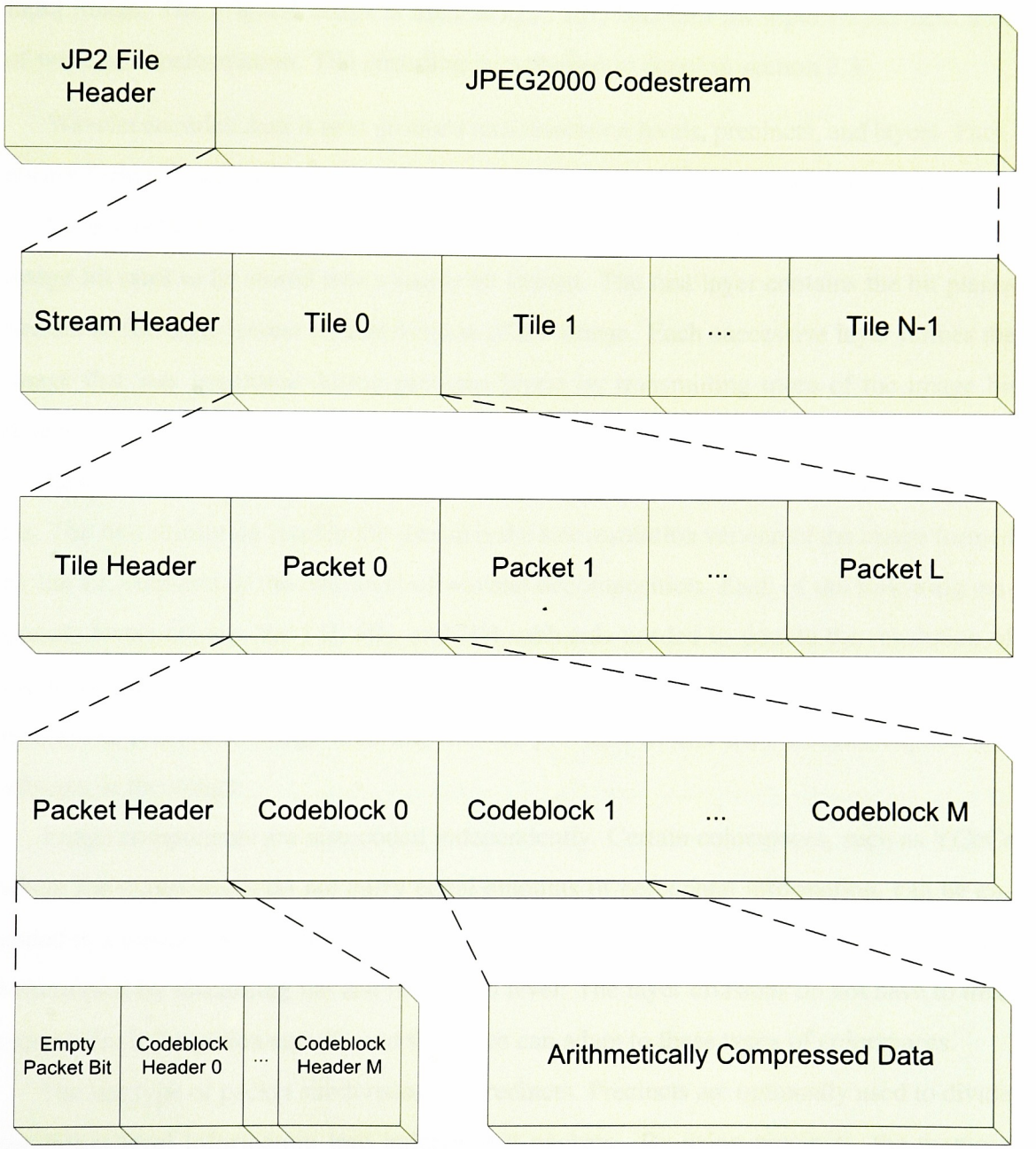

Figure 2.1: Components of a JPEG2000 Compressed Image

is a two-dimensional separable operation that takes an input image LL_{d-1} and produces four subband images LL_d, HL_d, LH_d, and HH_d that are each half the resolution of the input image. The original image is used as LL_0. LL_d becomes the input for the next level of wavelet transformation. Tile encoding is explained in detail in section 2.3.

Wavelet encoded data is next grouped into resolution levels, precincts, and layers. Packets are formed from each specific tile, layer, component, resolution level and precinct.

Layers within the JPEG2000 codestream are partitions that allow for multiple specific image bit rates to be stored into a single bit stream. The first layer contains the bit planes needed to form the lowest bit rate version of the image. Each successive layer refines the image that was generated during previous layers by transmitting more of the image bit planes.

Resolution levels resulting from the wavelet decomposition are coded in separate packets. The first resolution level in the stream is the low resolution version of the image formed by the LL-subband of the Nth level of wavelet decomposition. Each of the following resolution levels contain the LH, HL, and HH subbands needed to double the resolution of the image. By creating this partitioning of the codestream, decoders that are interested in a lower resolution image than the encoder provided do not have to fully expand and subsample the image.

Image components are also coded independently. Certain colorspaces, such as YCbCr where the components do not carry equal amounts of perceptual information, can be encoded in a manner that reflects their contributions. Chrominance channels can be implicitly subsampled by discarding the last resolution level. The layer divisions do not have to treat component information equally and therefore can adapt to these types of colorspaces.

The last type of packet subdivisions is precincts. Precincts are optionally used to divide resolution level information into independent packets. By using precincts, the memory requirements of the program can be reduced and provide another level of error resiliency.

Multiple components, precincts, and layers are optional features of the standard that were not incorporated into the DSP codec that was developed, thus packets within this

implementation will represent a resolution level within a tile. Using this format, for an N-level wavelet decomposition, N+1 packets are formed. The first packet is the LL subband from the final transform level (or the original image for a 0-level transform). Subsequent packets each contain the HL, LH, and HH subbands of data which, when combined with the current data available, represent the LL subband of the next higher resolution level. For a complete description of JPEG2000 packets, see section 2.4.

Packets are next divided into independently encoded codeblocks. The packet header gives information on which codeblocks, if any, are included in the packet. For each included codeblock, the packet header also gives information on the length of the codeblock data, the number of coding passes included, and the number of bit planes starting from the most significant bit that are entirely zeros.

Codeblocks are arithmetically coded one bit plane at a time using an MQ-coder. The bit for each codeblock coefficient in that bit plane is coded in one of three coding passes, namely the Significance Propagation Pass, Magnitude Refinement Pass, and Cleanup Pass. When the first non-zero bit of a coefficient is encoded, that coefficient is said to become significant. Coefficients that are already significant are coded in the Magnitude Refinement Pass, those with significant neighbors are coded in the Significance Propagation Pass, and all others are coded in the Cleanup Pass. The encoded codeblock stream may also be truncated to an arbitrary length. Lossy JPEG2000 compression is achieved by omitting some or all of the bit planes of the wavelet encoded data for particular codeblocks. For details on codeblock and arithmetic coding, refer to section 2.5.

2.2 Stream Header

The stream header is organized as a collection of marker segments. These markers consist of a 16-bit marker code, 16-bit length of marker segment, and marker segment data. The stream header contains all of the information needed to reconstruct the compressed code stream, including the image and tile sizes, number of components, quantization, and

wavelet transform parameters.

2.3 Tile Encoding

2.3.1 Tile Header

Tile headers use the same marker segment format as the main header. The only required field in the tile header indicates the length of the tile data and the tile index the data is associated with. Optionally, the tile header may override most settings from the stream header.

2.3.2 Tile Data

JPEG2000 images are divided into one or more tiles. The first step in tile encoding is to perform a DC level shift if the input data is to be interpreted as unsigned samples. For 8-bit per sample unsigned components, 128 is subtracted from each sample to produce values in the range -128 to 127.

The level shifted values are then processed row-wise by a 1-D wavelet transform, followed by a column-wise 1-D wavelet transform of the transformed row coefficients. JPEG2000 defines two wavelet transforms that may be used. A low-complexity LeGall (5,3) integer transform is used for reversible transformations and a Daubechies (9,7) floating point filter for irreversible transformations. Prior to applying the wavelet transform to input signal X, a symmetric periodic extension is performed to pad the endpoints so that calculations involving values outside the signal range are filled appropriately. The values to use as padding to form X_{ext} from the input signal X are illustrated in Figure 2.2. The transforms are defined as a series of lifting steps as shown in equations 2.1 through 2.8. Y represents the output of the transform and X_{ext} is the periodically extended input signal. The coefficients $\alpha, \beta, \gamma, \delta$, and K for the irreversible wavelet transform are defined in Table 2.1. The lifting steps are applied iteratively to the odd-indexed and even-indexed values. Each lifting step uses the output from the previous two steps as its input.

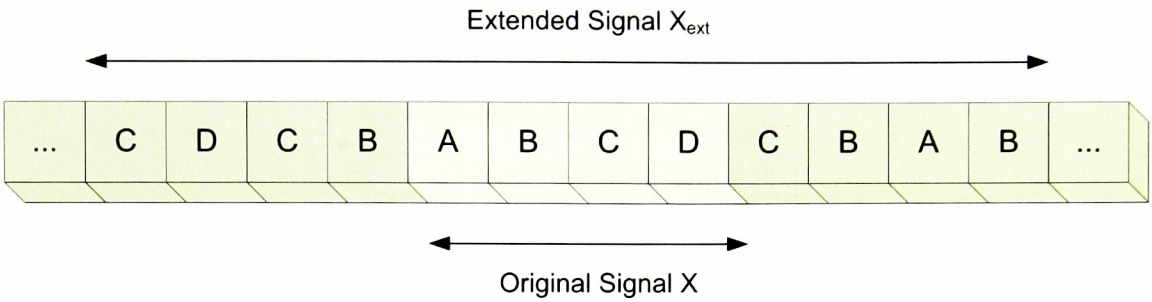

Figure 2.2: Periodic Extension for Wavelet Transformation

5-3 Reversible Wavelet Transform Lifting Steps:

$$Y(2n+1) = X_{ext}(2n+1) - \lfloor \frac{X_{ext}(2n) + X_{ext}(2n+2)}{2} \rfloor \tag{2.1}$$

$$Y(2n) = X_{ext}(2n) + \lfloor \frac{Y(2n-1) + Y(2n+1) + 2}{4} \rfloor \tag{2.2}$$

9-7 Irreversible Wavelet Transform Lifting Steps:

$$Y(2n+1) = X_{ext}(2n+1) + \alpha(X_{ext}(2n) + X_{ext}(2n+2)) \tag{2.3}$$

$$Y(2n) = X_{ext}(2n) + \beta(Y(2n-1) + Y(2n+1)) \tag{2.4}$$

$$Y(2n+1) = Y(2n+1) + \gamma(Y(2n) + Y(2n+2)) \tag{2.5}$$

$$Y(2n) = Y(2n) + \delta(Y(2n-1) + Y(2n+1)) \tag{2.6}$$

$$Y(2n+1) = KY(2n+1) \tag{2.7}$$

$$Y(2n) = (1/K)Y(2n) \tag{2.8}$$

After the wavelet transform is performed in each direction, the data is separated into four subbands. To define the partitioning, indices (iRow, iCol) will be assigned to the rows

Symbol	Approximation
α	-1.586134342059924
β	-0.052980118572961
γ	0.882911075530934
δ	0.443506852043971
K	1.230174104914001

Table 2.1: 9-7 Irreversible Wavelet Transform Coefficients

and columns of the transformed data respectively, where the upper left sample is (0, 0). The samples with even iRow and even iCol values are the LL subband, samples with even iRow and odd iCol values are the HL subband, samples with odd iRow and even iCol values are the LH subband, and samples with odd iRow and odd iCol values are the HH subband.

The subbands of transformed data exhibit these properties:

- LL subband is half resolution image of the input
- LH subband contains vertical edge information
- HL subband contains horizontal edge information
- HH subband contains information where both horizontal and vertical edges exist

The LL subband is used as the input to the next level of wavelet transform and the process is repeated N times where N is the number of wavelet decomposition levels used for the image. Figure 2.3 shows a representation of an image in the wavelet domain. From this image it can be seen that the majority of the image data details are now concentrated in the upper levels of the wavelet data.

After the wavelet domain representation is produced, it is divided into packets of data at the same resolution level. Figure 2.4 shows this partitioning.

Tiles are encoded independently to provide error resiliency, reduce memory requirements, and provide random access within the image. Tiles are given an index that identifies the location of the tile data within the complete image. The indices start at 0 and progress

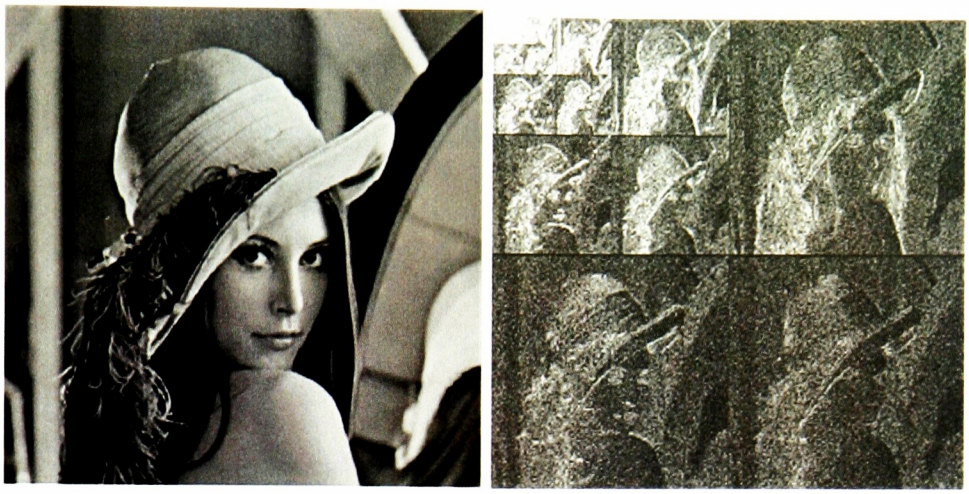

Figure 2.3: Spatial and Wavelet Domain Representations of Lena

in raster order, left to right, top to bottom. Each tile has a tile header that includes, among other information, the tile index and the offset to the end of the tile. Using this information, decoders can chose to decode only portions of the image by bypassing unnecessary tiles.

2.4 Packet Encoding

2.4.1 Tag Trees

Packet data for codeblock inclusion and insignificant bitplanes is coded in tag tree format. Tag trees are hierarchical structures that represent a two-dimensional array of nonnegative integers. The tag tree consists of N levels, where N is defined in equation 2.9.

$$N = \lceil \max(\log_2(rows), \log_2(cols)) \rceil \tag{2.9}$$

The highest level is the original two dimensional array of data to be encoded. The next level divides this high level in 2x2 squares. The minimum of each of these two by two squares is used as the entry for the next level node. Elements that are outside the original two dimensional arrays are treated as infinite. This process is applied iteratively until a level with only one node is reached. This structure is used to take advantage of adjacent

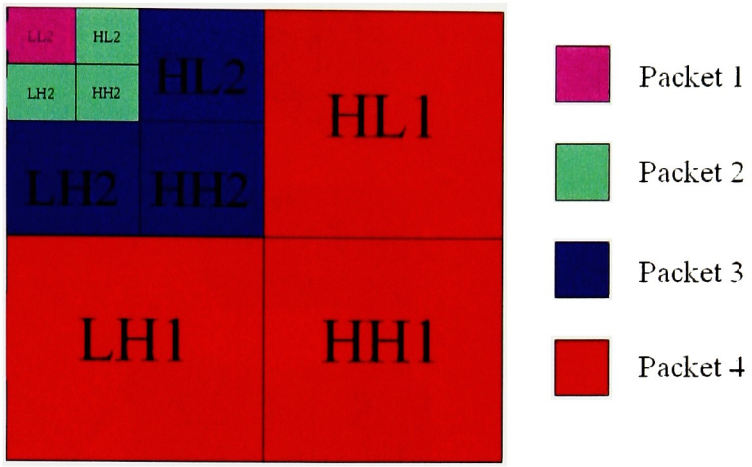

Figure 2.4: Packet Grouping of Wavelet Data

codeblocks having similar features. Figure 2.5 depicts the data that is contained in a tag tree structure.

To encode a node, the difference between that node and its parent node is computed. This number of '0' bits are needed, followed by a '1' bit. Decoders compute the value of the leaf nodes by initializing a count to zero and following the tree from the root to the leaf counting the number of '0' bits that were encoded for each node. When adjacent values are within a small range, the number of bits needed to encode the values is effectively reduced.

2.4.2 Packet Header

Using header information, the dimensions for the packet and the number of codeblocks that are contained within the packet are known. The packet starts with information that is needed for each codeblock grouped into a packet header. Information that is signaled in the header includes zero length packet, codeblock inclusion, number of insignificant most significant bit planes, number of coding passes for each codeblock in this packet, and the length of the codeblock data. The packet bitstream uses bit stuffing to avoid mistakenly including marker codes within the bitstream. Bits are inserted into the stream from MSB to LSB and if a byte is 0xFF, a zero bit is inserted into the MSB of the next byte before continuing.

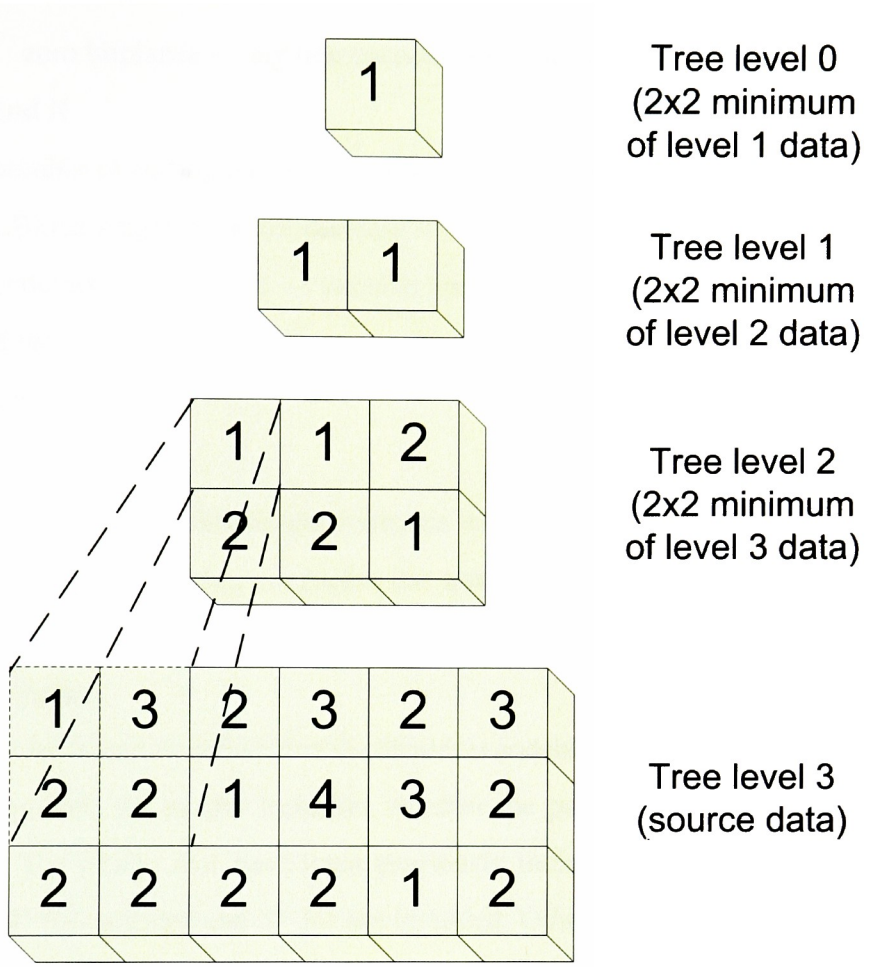

Figure 2.5: Tag Tree Data Representation

The following pseudo-code shows the data that is included in the packet header. The elements of the pseudo-code are later discussed in detail.

Non-zero packet length ← one bit
if Non-zero packet length **then**
  for all subbands in this packet **do**
    for all codeblocks in this subband in raster order **do**
      if the codeblock has been previously included **then**
        Codeblock inclusion ← one bit
      else
        Codeblock inclusion ← tag tree encoded layer of first inclusion

zero bitplanes ← tag tree encoded zero bitplane information

end if

number of coding passes ← variable length code

LBlock length indicator increase ← N+1 bits

codeblock data length ← variable length code

end for

end for

end if

The first bit of the packet header indicates whether any information is included in this packet. If the value is a one, the packet has a non-zero length. If the value is a zero, no codeblocks are included in this packet, the stream is padded out to a byte boundary and the packet is complete.

The next information included in the header is codeblock inclusion. For each codeblock within the packet, the header indicates whether the packet includes information for the codeblock. For blocks that have been previously included a single bit is used to signal inclusion, '1' for included and '0' for not included. Otherwise, this information is encoded in a tag tree structure representing the layer in which the codeblock first appears. The tree leaf node for the codeblock is traced back to the root and all unencoded nodes along the path from the root to the leaf are added to the header bitstream. Any bits that are encoded for a node are not repeated for other children of that node or in following layers. Inclusion tree encoding terminates either when it is first known that the codeblock of interest is not included or the leaf node has been encoded. If the codeblock is included, the remaining header information for that codeblock is encoded.

If this is the first time that the codeblock is included, zero bitplane information is next encoded from a tag tree format. Zero bitplane information is used to determine what bit position the first coding pass will be operating on. Unlike the inclusion tag tree, the zero bitplane tree always encodes until the leaf node is encoded. Subsequent packets that include this codeblock will use the number of coding passes already encoded to determine where

Number of Coding Passes	Codeword in Packet Header
1	0
2	10
3	1100
4	1101
5	1110
6 – 36	111100000 – 111111110
37 – 164	1111111110000000 – 1111111111111111

Table 2.2: Variable Length Codes for Coding Passes

to start.

The number of coding passes is next encoded in the header. Variable length codewords are used to signal the number of coding passes. Table 2.2 shows the codewords used.

The final two fields - LBlock length indicator increase and codeblock data length combine to encode the number of bytes of arithmetically encoded data for this codeblock. Codeblock data length reads N bits from the input where N is defined in equation 2.10.

$$N = LBlock + \lfloor \log_2(coding\ passes) \rfloor \tag{2.10}$$

These raw bits are treated as an unsigned value that is the number of raw bytes. Since N may not be large enough to hold the number of bytes, the data length is preceded by a LBlock length increase. LBlock is a state variable of the codeblock that is initialized to 3. The LBlock length indicator increase field is encoded by inserting N '1' bits where N is the amount to increment LBlock by, followed by a single '0' bit.

2.4.3 Packet Data

The packet data consists of the data that was signaled by the codeblock data length for each included codeblock. The data is included in the same raster order that was used for the packet header. Each of the codeblocks are encoded independently using the process shown

in the following section.

2.5 Codeblock Encoding

The codeblocks are arithmetically encoded one bitplane at a time using an MQ-coder. First, the block of wavelet encoded data is examined to find the most significant plane where a coefficient whose absolute value has a '1' bit in that position. The maximum number of bit planes is also computed from values signalled in the main header. The difference between the maximum plane and the first non-zero plane is stored for later coding in the packet header.

2.5.1 Codeblock State Variables

A state variable structure is used during coding having one entry for each pixel location in the codeblock. Each entry has variables for sign, significance, previous significance passes (LSig), and propagation. JPEG2000 codeblock coefficients use a sign-magnitude representation and the sign variable will be used for holding the sign. Each coefficient has an associated significance state. Prior to coding, this state is initialized to 'insignificant'. A coefficient becomes significant once the leftmost '1' bit in the coefficient has been arithmetically coded. LSig is used to aid in forming contexts for the arithmetic coder and will be discussed in later sections. Propagation is used communicate between coding passes. Both LSig and propagation are initialized to false.

2.5.2 Context Modeling

Context modeling is used to make accurate estimations of whether a bit to encode in a given situation is a '1' or a '0'. When encoding a bit, the states of the adjacent coefficients are used to form a context. The JPEG2000 specification [17] defines nineteen context states based on the value being coded, and the significance and sign of neighboring values. The MQ-coder uses the context and the bit to add to the codestream, then refines the prediction

for the given context for later use. In the following sections, when a bit is encoded, the rules for generating the context in that situation will be given. For the full details on context modeling, refer to Annex D of [17].

2.5.3 Coding Passes

Each bit of the coefficient is encoded in one of three coding passes, namely the Significance Propagation, Magnitude Refinement, and Cleanup Passes. The first non-zero plane is coded in the Cleanup pass and subsequent planes are processed by the Significance Propagation pass, Magnitude Refinement pass, and Cleanup pass in that order.

Coding passes operate on the codeblock in strip-based scan order. The scan starts at the top left position and scans downward for the first four values. The first four values from the second column are then scanned, followed by the third column and so on until the right side of the codeblock is reached. The scan then moves to the fifth row of the first column and again scans a four-line strip. This progression is repeated until all of the values in the codeblock have been scanned. Figure 2.6 shows an example scan order for a block that is 8 values wide.

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31
32	...						
...							

Figure 2.6: Strip Based Scan Order

Significance Propagation Pass

The Significance Propagation pass scans the codeblock and encodes each coefficient that is insignificant and has significant neighbors. If the coefficient being coded has a '1' bit in the current plane, the sign bit is encoded and the coefficient is now considered significant. The

propagation state flag set to TRUE for each value that is encoded in this pass to indicate that this position should not be coded in the other two passes for this bit plane.

Context modeling for the value bit is performed based on the significance state of adjacent values. The number of horizontally, vertically, and diagonally adjacent significant values are calculated. These three values, along with the current subband that the codeblock belongs to, are used in a table lookup and one of nine possible "New Significance Contexts" is selected.

Sign bit context modeling is more complex. First, context contributions are formed based on the significance and sign of adjacent horizontal and vertical values. These context contributions are then used to determine which of five possible "Sign Contexts" is used. The context contributions are also used to determine the predicted sign value. The bit to encode for the sign is determined by selecting the sign bit, and comparing it with the predicted sign. A '0' is encoded when the prediction is correct, and a '1' is encoded when it is incorrect.

Magnitude Refinement Pass

The Magnitude Refinement pass scans the codeblock and encodes the coefficients that are significant and were not coded in the Significance Propagation pass. One of three "Refinement Contexts" is used in the magnitude refinement pass. To determine the context of the bit, a flag (LSig) is kept for each value to indicate whether this value had previously been coded in the magnitude refinement pass. LSig is initialized to '0' and is set to '1' after a bit has been coded in the magnitude refinement pass for that coefficient. When LSig is '1', the first of the refinement contexts is used. Otherwise, the decision of which of the remaining contexts to use is determined by whether or not there are significant adjacent values.

Cleanup Pass

The Cleanup pass has two parts. In the first part, the contexts of the four vertical values in a scan strip are examined. If all of the neighbors of all of the values are insignificant, a bit is

encoded to indicate if any of the values will become significant in this pass. If a value will become significant, two bits are encoded to indicate the index of the first value such value. The first of these bits is encoded using the "Run-length Context" and the last two use the "Uniform Context".

In the second part all values that haven't been coded in the significance propagation pass, magnitude refinement pass, or part one of the cleanup pass are encoded. Adjacent values are used for context modeling. For values that become significant, the sign bit is also encoded. Values encoded in the second part use the same contexts as the significance propagation pass.

2.5.4 Arithmetic Encoder

The arithmetic encoder for JPEG2000 is a multiplication-free context-based adaptive binary arithmetic encoder. The block diagram of the arithmetic encoder is shown in Figure 2.7. A context index and value bit are taken as input. The more probable symbol (MPS) and current probability estimation state are loaded from the prediction for that context. The probability estimation state gives the probability of the less probable symbol (LPS), and information to adapt the estimation based on the value being coded. Values are encoded using recursive interval subdivision.

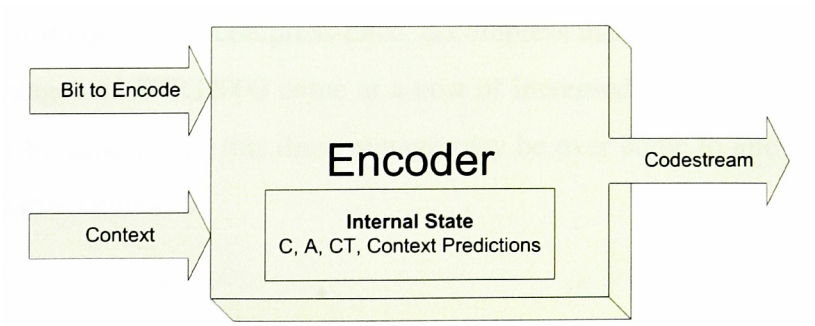

Figure 2.7: Arithmetic Encoder High Level Block Diagram

Three state registers, C, A, and CT, are kept by the encoder representing the code register, current interval, and count of available C-register bits respectively. To encode an LPS, A is reduced to the LPS probability estimate value (Qe). An MPS is encoded by

subtracting Qe from A and adding Qe to C. The interval A is initialized to 0x8000, which is equivalent to 0.75, and normalized when it falls below 0x8000 by left shifting by 1 bit. Using this method, A remains in the range $0.75 \leq A < 1.5$. Because the probabilities are based on a range of 1.0 and A is approximated, the LPS range may be larger than the MPS range. When this situation exists, the ranges are switched so that the MPS range remains larger than the LPS range.

The LPS probability estimate Qe is generated from a 47-entry state table. Each context contains a pointer into this table. Each state consists of the Qe value, the state transitions to perform if an LPS or MPS are encoded in this state, and a flag to signify if the predicted value changes when an LPS is encoded. By using this state machine, the probability estimates are able to use past information to predict future data.

2.6 Summary

The JPEG2000 standard is a very versatile format that offers options for lossy and lossless compression, up to 38 bits per component, and up to 16,384 image components. The image quality at very low bit rates is far superior to other existing standards, and regions of interest can be specified to further improve image quality. The format also allows progressive transmission of images and a compress-once, decompress many ways approach.

The advantages of JPEG2000 come at a cost of increased computational complexity. As processors become faster, this disadvantage may be over come to allow JPEG2000 to gain widespread acceptance.

Chapter 3

The Hardware Platform

In this chapter the hardware platform that the JPEG2000 codec implementation in this thesis was designed for is introduced. Figure 3.1 provides a high-level view of the system. Each component of the platform that is used by the codec will be examined.

3.1 Host System

The platform being used is built around a Sun Ultra 60 machine running Solaris 8. This system has dual 450 MHz processors and 512 MB of RAM. A 66 MHz PCI bus is used to interface with the acceleration board.

3.2 Acceleration Board

Residing on the 66 MHz PCI bus is the acceleration board. The processing on this board is performed by 4 Equator Technologies, Inc. BSP-15 digital signal processor chips [12]. Each BSP-15 is interfaced with its own 64 MB of RAM on the PCI board. The PCI interface allows the host system to broadcast messages and data to all BSP-15 processors. Receiving information back from the board is achieved by sending an empty output buffer to the board and waiting for the BSP-15 that is designated as the board master to send an interrupt signifying that the information has been written to that buffer.

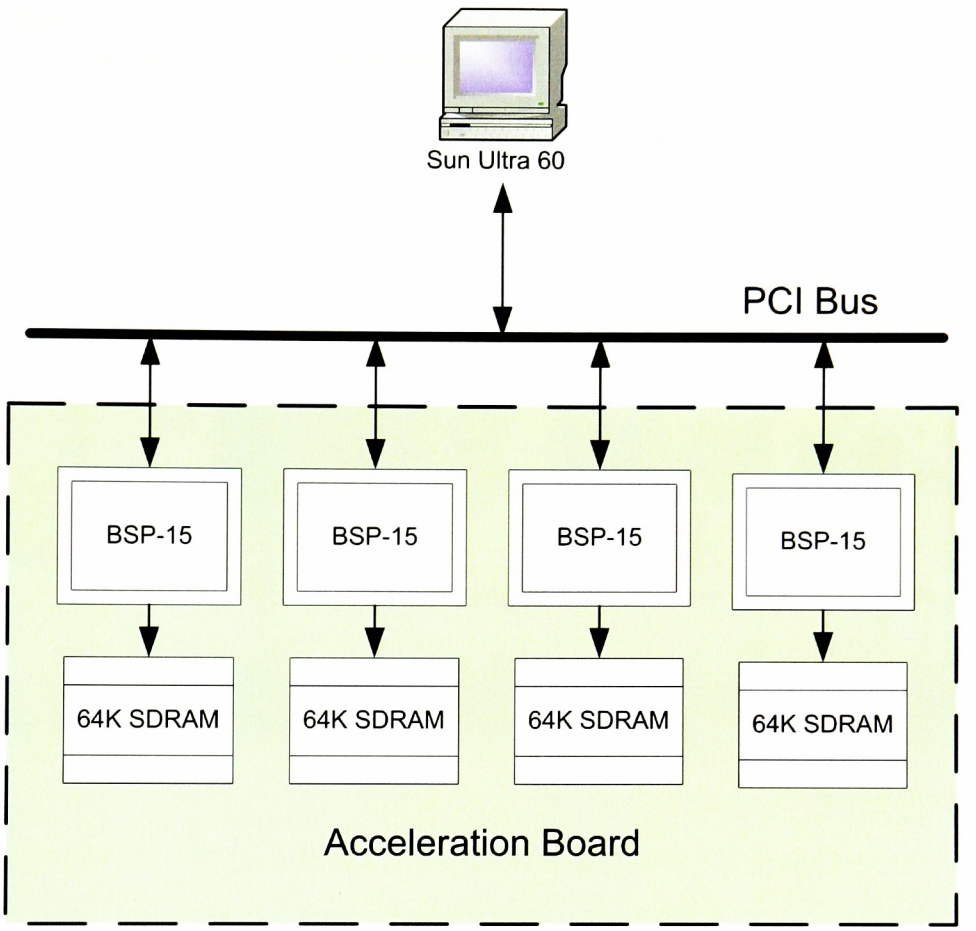

Figure 3.1: Platform High-level View

3.3 BSP-15 Digital Signal Processors

The Equator Technologies, Inc. BSP-15 is a 405 MHz, 0.15μm, TSMC technology digital signal processor. The low power requirements of these processors allow the acceleration board to operate on the power provided by the PCI bus without additional power supply connections. The integrated peripheral interfaces being used are the 33 MHz/66 MHz PCI bus interface and the glueless 64-bit/32-bit SDRAM controller. The BSP-15 was designed to support high-level programming techniques to shorten development time while still delivering full processor performance. Figure 3.2 shows the architecture of the BSP-15 processor [12].

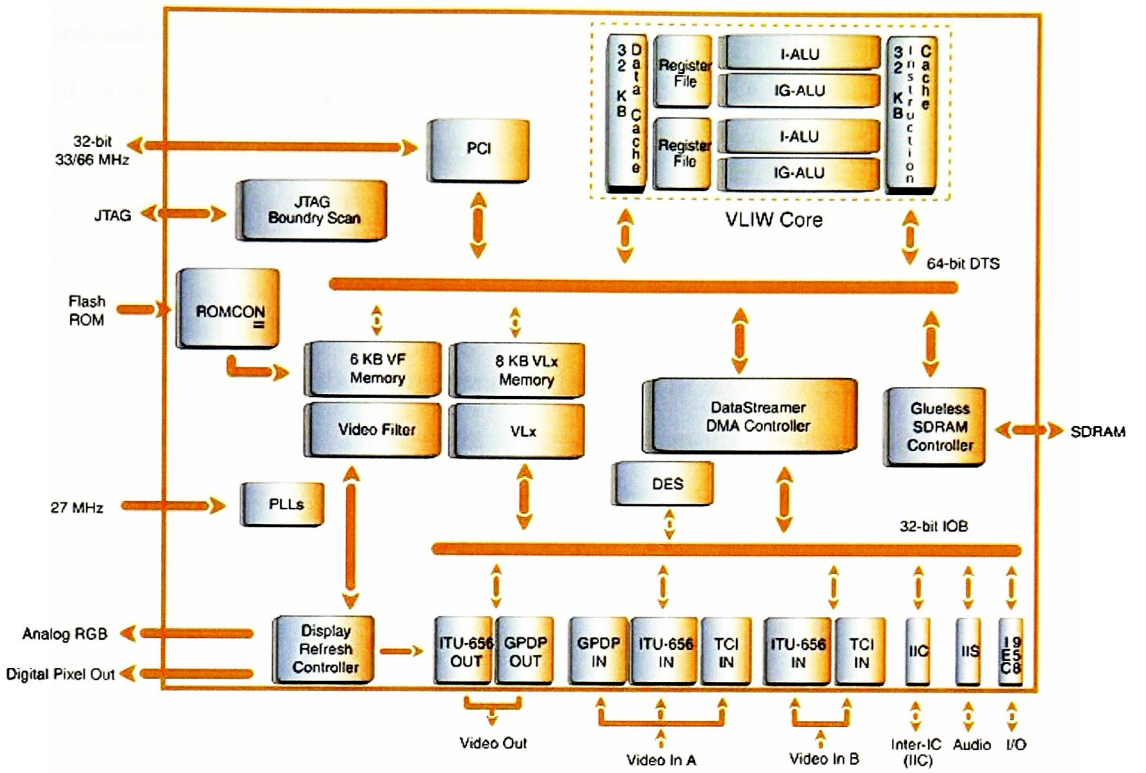

Figure 3.2: BSP-15 Architecture

3.3.1 VLIW Core CPU

The main processing of the BSP-15 is performed on a Very Long Instruction Word (VLIW) Core central processing unit (CPU). The VLIW core consists of four arithmetic logic units (ALUs). Two of the ALUs perform 32-bit integer operations(I-ALU), and the other two perform both 32-bit integer operations and 64-bit partitioned multimedia operations(IG-ALU). The core has 128 32-bit general purpose registers. Adjacent registers can be treated as a single 64-bit register.

Dividing the workload among the four ALUs is done statically at compile time. Programs are written in C, with intrinsic function calls that are mapped to the multimedia operations. Multimedia instructions are native single instruction multiple data (SIMD)

operations that must be specified by the programmer when writing the program. The 64-bit operands of these instructions can be partitioned into 8 8-bit values, 4 16-bit values, 2 32-bit values, or a single 64-bit value.

A wide range of multimedia operations exist to assist in computationally complex imaging operations. These SIMD multimedia instructions can be as simple as performing a 64-bit wide binary 'OR', and can get as complex as performing an inner product on 16 8-bit values. The inner product operation multiplies each of the 8 8-bit values in the first register by the 8-bit value in the corresponding location in second register, keeping the 16-bit result of each calculation. These eight intermediate results are then added together and the result is placed in the 32-bit destination register. Figure 3.3 shows the calculations that are performed during the inner product instruction.

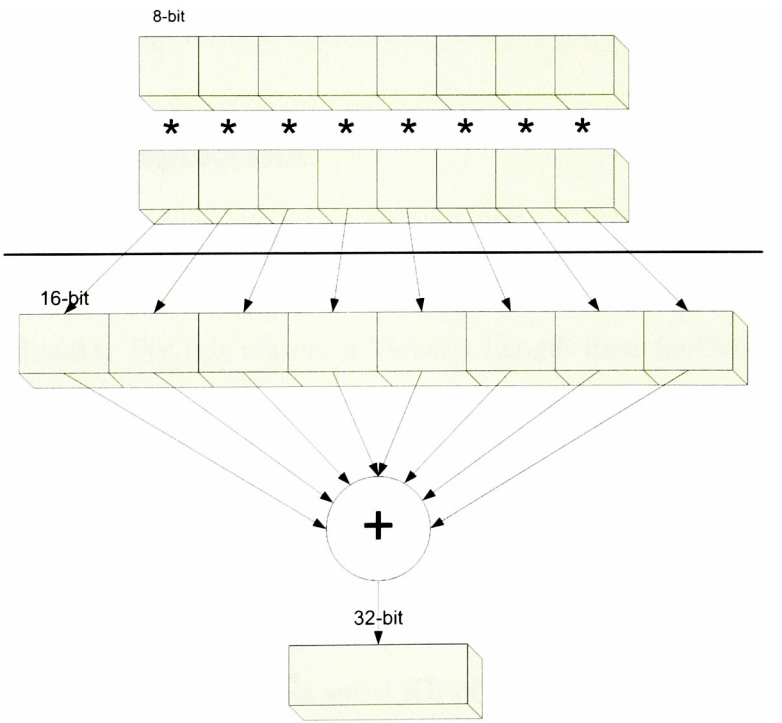

Figure 3.3: Inner Product Instruction Calculation

3.3.2 DataStreamer DMA Controller

The BSP-15 offers an on-chip DataStreamer Direct Memory Access Controller (DMA Controller). The DataStreamer provides an efficient method of copying data and has the ability to load portions of data into cache before it is needed by the program. The DataStreamer operates in parallel with the other resources on the BSP-15, allowing processes with high memory bandwidth requirements to "prefetch" the next block of data into cache while processing the current block.

The DataStreamer is programmed using a series of descriptors that combine to specify a memory-to-memory transfer. Descriptors specify the location and size of the memory to be copied, as well as whether the data should be put into cache and if the transfer should pause before continuing to the location specified by the following descriptor. The DataStreamer can also execute an interrupt routine when the transfer is completed.

3.3.3 VLx RISC Coprocessor

The VLIW core of the BSP-15 offers high performance on tasks that can take advantage of the parallel processing resources provided. The VLIW core is at a disadvantage when processing serial tasks. For this reason, a Variable Length Encoder/Decoder (VLx) coprocessor exists to offload tasks that would otherwise burden the VLIW core. The VLx is a reduced instruction set computing (RISC) processor with additional hardware support for accelerating Huffmann coding. The VLx may be programmed in C, and can also be programmed in assembly for optimal performance.

The VLx has 32 16-bit registers and all operations are performed on 16-bit signed values. There is 4 KB of memory for data and 4 KB of memory for instructions. The VLIW core may directly access VLx memory but not vice versa. VLx / VLIW core communication is typically handled through DataStreamer paths. The VLx has the ability to continue DataStreamer operations, thus signalling that processing is complete.

3.4 Summary

The platform being used offers several levels of parallelism. At the highest level there are operations happening on the Sun Ultra 60 and the acceleration board in parallel. On the board there is parallelism between the four BSP-15 processors. On each BSP-15 there is parallelism between the VLIW core CPU, the DataStreamer, and the VLx. On the VLIW core CPU, there are four ALUs that are processing up to four instructions per clock cycle. And finally, within the SIMD instructions, data is being operated on in parallel.

Chapter 4

JPEG2000 Codec Implementation

To take advantage of the hardware platform introduced in the previous chapter, the JPEG2000 implementation had to be designed specifically to use all available resources. ANSI-C versions of the encoder and decoder were used as a starting point and continuously adapted to improve performance. The goal of the implementation was to minimize the load on the Sun Ultra 60 and maximize the acceleration board utilization.

The codec is used by first loading the acceleration board program. This program will sit in a listening state until a command is received over the PCI bus, perform the compression or expansion, and return to the listening state. To invoke the JPEG2000 operations, a host program is used that processes the command parameters and calls the acceleration board.

4.1 Sun Ultra 60 Processing

4.1.1 Image Encoding

The main tasks that were performed on the Sun Ultra 60 were file I/O, allocating buffers, and calling the acceleration board for processing. For compression operations, the user specifies the raw input file, image dimensions, tile dimensions, decomposition levels, and output file. The input file is read into a memory buffer. An output buffer is allocated based on a worst-case compression guarantee. The dimensions are used to calculate the number of tiles in the image. A call to the acceleration board is then made with the following parameters:

- Input buffer
- Image height
- Image width
- Tile size
- Decomposition levels
- Maximum compressed tile size
- Output buffer - Output
- Compressed tile size for each tile - Output
- Header buffer - Output
- Header buffer size - Output

The acceleration board call blocks until the compression is complete. Upon returning, the data is gathered and written to the output file. The header buffer size tells how many bytes from the header buffer to write to the output file. Each tile is then written to the output file. Tiles are located in the output buffer at "maximum compressed tile size" boundaries and the size is given by the compressed tile size output parameter. This method of separating the tiles allows compressed data to be generated out of order without needing to know the sizes of other tiles.

4.1.2 Image Decoding

Processing for the decoder is much simpler. Input and output file names are specified. The image size information in the compressed input is parsed to determine the output buffer size needed. The input and output buffers are sent to the board. Upon returning, the output buffer contains the expanded image.

4.2 Acceleration Board Processing

4.2.1 Encoder

Upon receiving a compress command over the PCI bus, the BSP-15 processors on the acceleration board work together to process the image. All BSP-15 processors receive the same command and the processor identification number is used by each processor to determine its role in the compression. The processor with an identification number of '0' will be known as the board master. All values other than the parameters passed from the compress command will be stored in the processor local memory unless specified otherwise.

Tile Assignment

Since the JPEG2000 algorithm works on independently coded tiles, a tile was chosen as the task size that would be assigned to a BSP-15 processor. In early versions of the implementation, the tile assignments were statically assigned in a round-robin fashion. For most images, this approach produced nearly balanced loads among the BSP-15 processors. With static assignment, images may be created that cause the processor loads to be far out of balance. It was for this reason that a dynamic load balancing mechanism was designed.

The master keeps track of the next tile index that has not yet been assigned to a processor, a task assignment list in shared memory of the next tile that each processor should compress, and a flag in shared memory signalling that the image is completed. Processors check the task list for the next tile it is assigned. If it has not been assigned a tile, it will wait until either an assignment has been made or the image is complete. When an assigned tile is taken, the processor updates the master's task list to indicate that it can be assigned the next tile. The task assignment list is frequently checked by the master to reduce processors blocking for tasks and achieves dynamic load balancing among the BSP-15 processors.

Each processor initializes a header structure using the values that were passed as parameters. After initializing the task assignment list at the start of an image, the master encodes the JPEG2000 main header using the header structure data. The master then processes tiles

similar to the other processors with the only difference being that the master must update the task list.

Tile Processing

After a processor retrieves its assigned tile number, a DataStreamer path is opened to copy the image data from the PCI input buffer to local memory. While that transfer is being performed, buffers are allocated that will be needed during tile processing. For each level of wavelet decomposition, buffers are allocated that will hold the results of the wavelet transform. Copying the data to local memory serves two purposes. First, PCI reads of the image data are confined to this step where the delays can be overlapped with other operations. Second, the data in the local memory is aligned to allow processing using the 64-bit SIMD instructions.

Once the image data is in local memory and all of the buffers have been allocated, the wavelet transform is performed iteratively to generate the coefficients that will later be arithmetically encoded. A packet encoding procedure is then called for each decomposition level. The wavelet and packet encoding procedures are discussed more in following sections.

After packet encoding has been performed, each packet is represented by a buffer of compressed data and a buffer of header data. The tile header is then written to the PCI output buffer and a DataStreamer path is opened to write the packet header and data buffers to the PCI output buffer. While the output is being written by the DataStreamer, intermediate buffers that were allocated are now freed. After the DataStreamer completes, the packet data and header buffers are freed and the tile has been compressed. The processor returns from tile processing, requests another tile and repeats until the image compression is complete.

Wavelet Processing

The first step in wavelet processing is to expand the 8-bit input pixel values from the tile buffer into an array of 16-bit values in the packet buffer. While doing this expansion, the rows and columns are also shuffled such that the wavelet transform could be performed in place and the results would be located in the buffers for the appropriate subband.

The wavelet transform is next applied as a series of lifting steps, first to each column and then to each row. The 5-3 reversible transform uses two steps, Equations 2.1 and 2.2. The first step subtracts the average of the two adjacent values from each odd-indexed coefficients. The second step uses the results from the first step and the even-indexed coefficients. The average of the two adjacent step one results is divided by two and added to the even-indexed coefficient.

This process is then repeated on the LL-subband for the desired number of iterations.

Packet Processing

Packets are processed by first arithmetically encoding each codeblock in the packet. This step is the major bottleneck in JPEG2000. To improve the performance, the codeblock encoding implementations were designed for both the VLx coprocessor and VLIW core. Both versions are functionally identical and able to run in parallel. The VLx version offers slightly better performance since it was written in assembly code.

The first codeblock within the packet is started by the VLx. At the start of subsequent codeblocks, the VLx is checked first to see if processing on the previous codeblock has completed. If not, then the codeblock begins processing on the VLIW core. As the VLIW core processing is taking place, the VLx is periodically checked to see when it is ready to start another block. This continues until all codeblocks have been encoded.

Next, each codeblock's contribution to the packet header is written. The codeblock header segments are concatenated to form the packet header and the codeblock data segments are concatenated to form the packet data. Combining the header and data forms an encoded packet. A DataStreamer path is used to place the packets into the final tile output.

Because the tile output buffers were allocated before the compression, there is a worst-case compression ratio that must be met on each tile. To achieve a compression that is within the worst case compression guarantee, codeblocks are given a length cutoff constraint that will cause processing to stop once the limit is reached. The constraint is calculated based on the number of codeblocks remaining to be coded and the amount of space remaining until the constraint is exceeded. The codeblock constraint is not strict, but there is a strict constraint on packets in which the packet will be omitted if it is unable to fit into the space remaining.

Final compressed assembly is done by the Ultra 60. A buffer of noncontiguous tiles and the image header is sent back along with values telling how long each part is. The Ultra 60 performs a series of file write operations to result in the final JPEG2000 unwrapped codestream.

4.2.2 Decoder

Much of the implementation architecture is identical in the encoder and decoder. The order or the algorithm steps is reversed, the MQ-encoder is replaced with the MQ-decoder, and the lifting steps for the wavelet transform are inverted. The decoding sequence is as follows:

- Parse main header
- Parse tile header
- Decode packet header
- Decode codeblocks in the packet
- Perform inverse wavelet transformation
- Repeat for each packet
- Repeat for each tile

To avoid communication overheads, each processor parses the main header. The encoder synchronization and tile assignment methods are reused on the decoder. Context modeling in the codeblock encoding is performed the same way. The first major difference comes when replacing the MQ-encoder with the MQ-decoder. The general structure of the coder remains the same, but the algorithm for extracting an encoded bit performs different calculations and comparisons. Each bit that is decoded causes the same updates to be applied to the A and Qe registers that occurred in the encoder. For flowcharts detailing the MQ-coder algorithms, refer to Annex C of [17]. For the wavelet transform, the order in which the filters are applied and the lifting steps are reversed. The complimentary operations of each lifting step are applied in reverse order so that the transform is undone.

When sending the data back to the Sun Ultra 60, the recovered image data is placed into a single buffer that is in raster order. The only difference between this buffer and the input image that was originally compressed, is that the output buffer has the scanlines padded to a 8-byte boundary. This allows for all tile data to be written at 64-bit boundaries with multimedia intrinsics.

The features supported by the decoder are a superset of those provided by the encoder. Additional features such as the irreversible wavelet transform have been implemented. To test said features, the Kakadu reference software was used that provides complete coverage of features in the JPEG2000 format.

4.3 Synchronization Methods

4.3.1 Acceleration Board Calls

Synchronization must be performed at several levels within this implementation. When invoking the acceleration board processing, a blocking 'ioctl' function call is used. At the end of processing an image, the board master raises an interrupt to complete the ioctl function call. To ensure that all processors have finished, the BSP-15 processors block on

a barrier condition before this interrupt is raised.

4.3.2 BSP-15 Load Balancing

During processing, the BSP-15 processors must communicate to distribute the processing tasks (image tiles). This communication is managed by the board master. The processors communicate through an array in shared memory with two elements for each processor. These values are interpreted as 'the next tile to process', and 'all tasks assigned'. At the start of an image, each processor is responsible for initializing their respective elements and the master waits for initialization to complete before proceeding. The master keeps a variable corresponding to the next unassigned tile index number. Periodically, the master scans the 'next tile to process' array and for each processor that has a value representing 'waiting for a tile', makes the assignment and increments the unassigned tile variable. After the variable is incremented past the number of tiles in the image, the 'all tasks assigned' value is set to true. To retrieve tasks, the processors poll the 'next tile to process' value until either a task is received or all tasks have been assigned. Upon receiving a task, the tile index number is saved to local memory, the 'next tile to process' is set to 'waiting for a tile', and task received is processed. The board master obtains and processes tiles in the same manner as the other processors.

The dynamic load balancing of tiles has a small impact on images that have similar amounts of information throughout the image. On typical test images, around a 5% improvement was seen over static assignment of tiles. The improvement can vary, with the largest change coming when all tiles containing image information are assigned to a single processor and the others are idle. This dynamic load balancing scheme ensures that the most unbalanced that the processors can be is the time to process two image tiles.

4.3.3 VLx / VLIW Core Codeblock Assignment

Another area of synchronization is in the processing of codeblocks within packets. To synchronize the coordination between the VLIW core and VLx, a flag in VLx memory is periodically tested to detect VLx completion. The first codeblock is started by the higher speed VLx implementation. After preparing for the next codeblock, this completion flag is tested. If the VLx is still busy, the codeblock is started by the VLIW core. After each bit plane of the codeblock on the core has been processed, the VLx flag is checked again. When VLx completion is detected, the next codeblock is assigned to the VLx and begins processing. Using this method provides load balancing between VLIW core and VLx processing.

4.3.4 VLx Codeblock Processing

Synchronization is also needed within the VLx processing of a codeblock. The codeblock is unable to fit in VLx memory so a DataStreamer path is used to load the appropriate section of the block. Since the DataStreamer operates in parallel with the VLx, it is possible to pipeline the operations into VLx memory buffers A and B in such a fashion than buffer A is being processed while buffer B is being written out and reloaded and vice versa. Synchronization is needed to detect when the DataStreamer is clear to make another transfer and when data has completed being loaded. To implement this feature, DataStreamer interrupts are used. The interrupt is programmed to occur after selected transfers are completed and sets a flag in VLx memory indicating that the transfer has finished and another can be started. The use of DataStreamer interrupts also allows preprocessing to be performed on the VLIW core for some of the data being sent to the VLx. "VLx hint buffers" are computed based on the data that just finished being loaded before the flag is set for the VLx. These hint buffers are filled by performing binary AND and OR operations on each column of data within the strip being loaded and aid in the context determinations. As the VLx is processing and needs another buffer to be loaded, the flag from the DataStreamer

is cleared, the DataStreamer transfer is initiated, and the current VLx buffer is processed. When current buffer processing is complete, the flag from the interrupt is polled until it is set and the process is repeated.

4.4 Summary

Many areas of the implementation have been tailored specifically for the platform. The goal for all of these changes is to increase performance. The modifications range from dividing the work evenly among available BSP-15 processors, to using specialized SIMD instructions, to efficiently using processor resources. Each plays a part in creating a high performance JPEG2000 implementation.

Chapter 5

Results

To test the JPEG2000 codec implementation, a test suite of approximately 50 greyscale images were used. These test images are all 2552 pixels wide and 3300 scanlines tall. This corresponds to an 8.5" x 11" image at 300 dots per inch. The original color versions of the documents from which the test suite is based is available from [7]. The versions of the images used are available on the accompanying CD. These images represent a coverage of many types of images that are processed by a printer, including text, line art, drawings, and photographs.

5.1 Performance Comparison Methods

5.1.1 Speedup

To determine the performance of the multiprocessor DSP JPEG2000 implementation, comparisons are made with reference JPEG2000 implementations. The reference software was ran on the same Sun Ultra 60 that is the host of the multiprocessor DSP system. To calculate the speedup gained by using the multiprocessor DSP implementation, Equation 5.1 is used. Unless stated otherwise, execution times measure from the JPEG2000 processing time and omit the file reading and writing overheads.

$$Speedup = \frac{Reference\ software\ execution\ time}{DSP\ implementation\ execution\ time} \tag{5.1}$$

5.2 Reference Software

5.2.1 JasPer

JasPer is an open-source software implementation of the JPEG2000 standard written in C. [1]. Started as a joint effort between Image Power, Inc. and the University of British Columbia, this implementation is available as part of JPEG2000, Part 5 - Reference Software. This software is maintained by one of the primary authors, Michael Adams. Version 1.701.0 of the JasPer software, which was released February 8, 2004, was used for testing. In verbose mode, the encoding and decoding execution times are given by the program.

5.2.2 JJ2000

JJ2000 is a java implementation of the JPEG2000 standard. This codec was the result of a joint effort between EPFL, Canon Research France, and Ericsson that ended in September 2001 [11]. This implementation was also included in JPEG2000, Part 5 - Reference Software. Version 4.1 of this software was used. This software does not provide an execution time interface. Performance comparisons against the JJ2000 implementation will use the end-to-end execution time which includes file IO overheads.

5.2.3 Kakadu

Kakadu is a heavily optimized JPEG2000 implementation developed by David Taubman [24]. Taubman was also the principle author of the JPEG2000 Verification Model software and coauthor of [25]. Kakadu software is designed to provide quick execution time and low memory usage. Solaris Version 4.2.1 of this software was used, which was released June 24, 2004. Execution time information is retrieved by using the '-cpu 0' argument.

5.3 Lossless Compression

In the first test, the performance of the multiprocessor DSP implementation was measured with a varying number of BSP-15 processors enabled. This measurement allows the scalability of the implementation to be determined. Tables A.1 and A.2 shows the execution times for compressing each image in the test suite with the following parameters:

- Height = 3300, width = 2552
- Tiles = 256 x 256
- Codeblocks = 64 x 64
- 5 wavelet decomposition levels
- Reversible transformation
- All bit planes encoded

The information from Tables A.1 and A.2 is summarized in Table 5.1 for each of the source documents.

5.3.1 Multiprocessor DSP Implementation

In Tables 5.1, A.1 and A.2, the first two columns identify the original document and page number in [7] from which the image was generated. The next column shows the execution time (sec) when the compression is ran using only one of the BSP-15 processors on the acceleration board. The next two columns show the execution time from using two BSP-15 processors and the speedup vs. the single BSP-15 time. Speedup is calculated by dividing the single BSP-15 time by the multiple BSP-15 time. Finally, the last two columns show the execution time and speedup when using all four of the BSP-15 processors on the board. These tables show that the execution time of the multiprocessor implementation scales nearly perfectly with the number of processors used.

Document	Pages	1 BSP-15	2 BSP-15	Speedup	4 BSP-15	Speedup
ArtCatalog	8	9.296	4.706	1.976	2.419	3.843
Barnsley	4	9.849	4.974	1.980	2.570	3.831
Castle	1	10.662	5.379	1.982	2.786	3.827
Dac97	21	7.893	3.998	1.971	2.076	3.783
GardenBook	10	10.285	5.201	1.967	2.671	3.811
PoolHall	1	11.917	6.044	1.972	3.066	3.887
QpsAll	7	7.180	3.646	1.969	1.897	3.780
Relay	1	8.763	4.434	1.976	2.289	3.828
TwoYears	1	8.892	4.518	1.968	2.314	3.843

Table 5.1: Average BSP-15 Execution Time (sec) Summary for Lossless Compression

The measurements made for compression were then repeated for the expansion of the resulting JPEG2000 images. Tables A.3 and A.4 show the results of these timings. The expansion program ran slightly slower than the compression. The majority of the performance change results from the different processing that must be performed in arithmetic encoder and decoder. The information from Tables A.3 and A.4 is summarized in Table 5.2 for each of the source documents.

Document	Pages	1 BSP-15	2 BSP-15	Speedup	4 BSP-15	Speedup
ArtCatalog	8	9.315	4.730	1.969	2.452	3.799
Barnsley	4	10.748	5.457	1.970	2.868	3.749
Castle	1	11.457	5.796	1.977	3.020	3.793
Dac97	21	7.574	3.853	1.955	2.013	3.706
GardenBook	10	11.760	5.977	1.915	3.122	3.607
PoolHall	1	14.561	7.414	1.964	3.900	3.734
QpsAll	7	7.342	3.763	1.949	1.992	3.676
Relay	1	9.553	4.848	1.970	2.564	3.726
TwoYears	1	8.805	4.476	1.967	2.313	3.808

Table 5.2: Average BSP-15 Execution Time (sec) Summary for Lossless Expansion

5.3.2 Timing Breakdown

To determine which parts of the JPEG2000 implementation were accounting for the processing time, tests were ran that gathered timing information for the major components of the encoder. Table 5.3.2 shows the breakdown of the encoding time for the PoolHall image. Table 5.3.2 further divides the processing time for the Codeblock Encoding step. As these tables show, most of the complexity of the JPEG2000 lies within the encoding of codeblocks.

Processing Step	Percentage of Total Execution Time
Wavelet Encoding	2.5%
Codeblock Setup	11%
Codeblock Encoding	75%
Other*	11.5%

*Other consists of Header writing, PCI operations, synchronization, etc.

Table 5.3: Encoding Timing Breakdown

Codeblock Processing Step	Percentage of Total Execution Time
Context Modeling	11%
Arithmetic Encoding	22%
VLx Hint Calculation	16%
Other*	26%

*Other consists of determining coding passes, traversing the codeblock, etc.

Table 5.4: Codeblock Encoding Timing Breakdown

5.3.3 Reference Software Performance

Kakadu and JasPer Encoding

The same suite of images was next compressed with available reference software using the same compression options. The execution times of the Kakadu and JasPer implementations

are compared with the execution time of the multiprocessor DSP implementation using all BSP-15 processors on the acceleration board in Tables A.5 and A.6. The information from Tables A.5 and A.6 is summarized in Table 5.5 for each of the source documents. The average speedup of the DSP implementation is 2.7 over the Kakadu software and 26 over the JasPer software.

Document	Pages	BSP-15	Kakadu	Speedup	JasPer	Speedup
ArtCatalog	8	2.419	6.433	2.646	61.309	25.335
Barnsley	4	2.570	6.910	2.700	63.908	24.911
Castle	1	2.786	7.160	2.570	67.380	24.187
Dac97	21	2.076	5.219	2.524	52.597	25.887
GardenBook	10	2.671	7.839	2.976	67.872	26.672
PoolHall	1	3.066	11.190	3.650	85.140	27.769
QpsAll	7	1.897	5.534	2.951	51.423	27.366
Relay	1	2.289	7.230	3.158	61.270	26.766
TwoYears	1	2.314	6.280	2.714	59.020	25.508

Table 5.5: Average Reference Software Execution Time (sec) Summary for Lossless Compression

Kakadu and JasPer Decoding

Tables A.7 and A.8 display the Kakadu and JasPer decompression performance for losslessly compressed images and is summarized in Table 5.6. The source images for this test were generated using the compression parameters described earlier in this section. The decoding performance of these implementations was roughly the same as the encoding performance.

JJ2000 Encoding and Decoding

The comparisons with the JJ2000 reference software had to be measured in a slightly different fashion. Since the JJ2000 codec does not provide a timing measurement interface, the unix 'time' command was used to measure the end-to-end execution time of a process

Document	Pages	4 BSP-15	Kakadu	Speedup	Jasper	Speedup
ArtCatalog	8	2.452	4.464	1.799	61.309	25.167
Barnsley	4	2.868	5.048	1.765	63.908	22.304
Castle	1	3.020	4.950	1.639	67.380	22.309
Dac97	21	2.013	3.273	1.617	52.597	27.509
GardenBook	10	3.122	6.010	1.956	67.872	23.764
PoolHall	1	3.900	9.790	2.510	85.140	21.832
QpsAll	7	1.992	3.953	1.990	51.423	26.283
Relay	1	2.564	5.460	2.129	61.270	23.895
TwoYears	1	2.313	4.280	1.851	59.020	25.522

Table 5.6: Average Reference Software Execution Time (sec) Summary for Lossless Expansion

that read the input from a file, performed the compression, and wrote the output to a file. A similar measurement was done for the four processor DSP implementation. The results of this test for both lossless compression and expansion are shown in Tables A.9 and A.10 and is summarized in Table 5.7. As these tables show, JJ2000 is much slower during expansion than it is for compression. The process typically showed 85-90% CPU utilization during compression and only 20-25% CPU utilization during expansion. Perhaps this discrepancy is due to inefficient parsing of the compressed stream. The average speedup for compression is 8.2 and the average speedup for expansion is 22.

5.4 Summary

The addition of multiple processors to the DSP implementation caused the execution time to scale inversely proportional to the number of BSP-15 processors used. Based on these measurements, it is seen that JPEG2000 is very well suited for multiprocessor systems. There is a very low amount of redundant computation and communication overhead causing the speedup to be just short of the number of processors used.

Document	Pages	4 BSP-15 Compress	JJ2000 Compress	Speedup	4 BSP-15 Expand	JJ2000 Expand	Speedup
ArtCatalog	8	3.389	24.6	7.262	3.539	79.1	22.589
Barnsley	4	3.558	25.5	7.169	5.194	78.8	15.732
Castle	1	3.751	28.0	7.464	4.973	79.0	15.886
Dac97	21	2.989	23.8	7.958	3.318	73.5	23.831
GardenBook	10	3.701	32.9	9.025	4.430	84.3	20.977
PoolHall	1	4.305	36.0	8.362	5.340	107.0	20.039
QpsAll	7	2.855	26.1	9.184	3.255	71.6	22.819
Relay	1	3.317	31.0	9.347	3.716	79.0	21.258
TwoYears	1	3.249	27.0	8.309	3.406	78.0	22.900

* All time information in this table represents
total execution time including file I/O overhead

Table 5.7: Average JJ2000 Software End-to-End Time (sec) Summary for Lossless JPEG2000

The multiprocessor DSP implementation of the JPEG2000 codec was able to outperform all tested reference implementations running on a Sun Ultra 60. The open-source implementations - JasPer and JJ2000 - saw speedups of over 5 when compared to the single BSP-15 processing time. When all of the BSP-15 processors are enabled, this jumps to a speedup of 20. Kakadu was the best performing reference implementation. The multiprocessor implementation was able to double the execution speed of the Kakadu software.

The results discussed in this chapter focussed on 8.5" x 11" 300 DPI print resolution images that are divided into 130 tiles. The processing speed (in pixels per second) remains the same on lower resolution images or different tile sizes. The only provision is the tile divisions be sized such that there are an adequate number of tiles to evenly distribute the processing load among the BSP-15 processors. For example, a display resolution image (72 DPI) would take approximately 1/16th of the processing time and tiles would need to be reduced to approximately 64x64 to achieve good load balancing.

Chapter 6

Conclusions

The work described in this thesis produced a multiprocessor DSP implementation of the JPEG2000 standard. This implementation is competitive with existing implementations and identifies features of a platform that can be used to accelerate JPEG2000 operations. Using the information gathered, the number of Equator BSP-15 processors that is needed to achieve a desired processing throughput is known. This processor, in its current version, may not perform well enough to produce a practical platform for high speed JPEG2000 applications but does outperform the current software alternatives.

The observations seen on this thesis creates a profile for what may be considered an ideal JPEG2000 platform. First, the platform must mirror the parallelism that exists within the algorithm. Mapping tiles to processors provides a low communication overhead partitioning. SIMD instructions can efficiently handle the processing of wavelet transforms. It was also observed that codeblock processing accounts for approximately 75% of the execution time. Codeblocks are, by definition, independent of each other and would be helped by supplying coprocessors to divide and conquer this task. Within these coprocessors, the performance can be further improved by integrating hardware functionality for arithmetic coding.

6.1 Platform Analysis

6.1.1 Host System

For this implementation, there is essentially no load on the host Sun Ultra 60 system. This gives the opportunity for the host system to be running other programs such as server applications or printer system controllers. The CPU utilization of the host system was 0% for the DSP codec, whereas the utilization for the reference software with lower performance is over 80%. If the JPEG2000 codec is used within a system that requires other processing, the processing speeds measured in Chapter 5 will be slower due to processor utilization hitting its limit. By using the DSP JPEG2000 codec, the host system is going unused, but has the benefit of providing greater flexibility.

6.1.2 Acceleration Board

The acceleration board provides a compact multiprocessor environment. As was seen in the previous chapter, the performance of the JPEG2000 codec is able to scale nearly perfectly as the number of processors available is increased. The acceleration board adds four processors on a single PCI board. When compared with other multiprocessor configuration options, this setup offers lower power consumption, faster communication, and less bulk. The degree of parallelism possible in this setup is limited by the number of PCI slots available in the host system, but is great enough to achieve dramatic speedups over existing codecs.

6.1.3 Equator BSP-15 Processors

The Equator Technologies, Inc. BSP-15 digital signal processor offers features to excel in the performance of the wavelet transform and other computations within the standard. It does not give much help for performing the context modeling and arithmetic coding. This causes the codeblock coding time to consume approximately 75% of the execution time.

For this step, the BSP-15 software is programmed using instructions that are available in a general purpose processor.

A large advantage that the BSP-15 has over the comparable general purpose processor is the difference in power consumption. The entire four processor acceleration board uses under 25 Watts of power, whereas a 1700 Mhz Pentium 4 processor alone consumes 64 Watts of power. This low power consumption is what enables the multiprocessor environment to reside on a single PCI board.

6.2 Suggested Platform Improvements

6.2.1 Host System

Depending on the target application, a more powerful host system could be used. The host system could take an active role during the JPEG2000 processing by performing one of the processing steps such as the rate control computations. The host should have support for SIMD operations to help in computationally heavy steps.

The role of the host system may also move in the other direction. An implementation could be possible that did not require the assistance of a host system at all. This would be the approach needed to pursue the development of a JPEG2000 based digital camera. Such a digital camera would be easily extended to be Motion JPEG2000 digital video camera. Motion JPEG2000 files consist of a series of independently coded JPEG2000 images [15], removing the need for motion estimation that is a major step in other codecs such as MPEG.

6.2.2 Acceleration Board

The major changes for the acceleration board will stem from changes in the BSP-15 processor itself. If subsequent versions of the chip require less power, it may be possible to increase the number of processors on the board. If the processing power increases such

that the overhead of communicating over the PCI bus becomes a significant portion of the execution time, the PCI interface may have to be exchanged for an AGP or PCI Express interface. PCI Express would also increase the amount of power that could be supplied to the acceleration board. This may enable more DSP chips to be added to the board.

Also, as mentioned previously, an acceleration board that can be removed from the Sun Ultra 60 and placed independently into a device like a digital video camera would present many new opportunities. Power usage would still be a concern, but the limitation of staying below the power available over the PCI interface would be removed. This opens the door for acceleration boards that have many more DSP processors on them to meet the demands of a larger range of products.

As with all imaging applications, the amount of data that is processed is very large. The BSP-15 reduces the memory latency impact by providing the DataStreamer that fetches the data from SDRAM into cache.

6.2.3 BSP-15 Processor

VLIW Core

The current performance bottleneck of the JPEG2000 implementation is codeblock encoding. To address this issue, hardware support should first be integrated to perform the arithmetic encoding and decoding. I envision this as being similar to a feature that already exists within the VLx that assists Huffmann coding. Pipelined hardware implementations of the MQ-coder exist that would be well suited to provide this functionality [14].

From table 5.3.2, the codeblock coding time would still account for over half of the processing time and be the performance bottleneck. To further reduce the bottleneck, additional general purpose coprocessors could be added. These coprocessors would be similar to the VLx, offering more computing power for tasks that are highly serial in nature.

Several areas could also benefit by supporting 128-bit or 256-bit wide SIMD operations. The wavelet transform in particular operates on sets of data that are as wide as the tile.

Wider operations would reduce the number of iterations of the loop needed to process the transform.

VLx Coprocessor

Several obstacles were encountered when implementing the VLx programs for codeblock operations. The first of which was the small amount of VLx memory available. The VLx can only hold four kilobytes of data and the codeblocks being operated on were over eight kilobytes alone. The complex DataStreamer transfer that needed to be set up could have been avoided had more memory been available. Ideally the VLx memory should have been at least sixteen kilobytes for the codeblock encoding and decoding functions.

Another VLx programming obstacle was encountered when developing the MQ-coder functions. The algorithm calls for 32-bit unsigned registers to be used to keep the MQ-coder state. The VLx however, only has operations for 16-bit signed values. VLx memory requirements could also have been reduced by allowing 8-bit values.

6.3 Future Work

6.3.1 Software

New Features

The JPEG2000 codec described in this work implemented the fundamental features of the JPEG2000 algorithm. There are still several options that are not supported. Future work may build upon this base to create a full implementation of the specification. Major features that are currently unsupported include:

- Region of interest handling
- Lossy compression rate control

- Multiple component images

The addition of these features will have little impact on the handling of features that are already supported.

Hardware Changes

As with any program that is tailored for a specific platform, the code will need to be updated if and when the components of the platform change. As future versions of the BSP-15 processor incorporate features that would improve the performance of this software become available, this software should be updated accordingly.

New Algorithm Optimizations

Several approaches to efficient programming of codeblocks have been examined recently [26] [22] [6]. Each of these research efforts were able to reduce redundant or unnecessary calculations to produce an improved codeblock encoding scheme. As new approaches are developed and published, they should be compared against the current implementation. Techniques that give significant performance increases should be adopted and incorporated into the codec.

6.3.2 Hardware

Latest Semiconductor Technology

One of the largest factors in determining the cost, size, power consumption, and clock rate of a processors is the transistor technology used. The Equator BSP-15 chip currently uses 150nm Taiwan Semiconductor Manufacturing Company (TSMC) transistor technology. The current state of the art process is 90nm technology, which is already being used in DSP processors produced by Texas Instruments. To stay competitive, Equator Technologies, Inc. must move to smaller transistor sizes. This will enable the clock rate to be increased, thereby increasing performance of the JPEG2000 codec.

Arithmetic Coding Support

JPEG2000 could benefit greatly by the addition of hardware arithmetic coder support. By designing a module that performs the MQ-coder functionality of annex C in [17], the JPEG2000 implementation could virtually eliminate the 22% of the time for arithmetic coding seen in Table 5.3.2. There has been research performed in this area [31] [14] and the next step is to incorporate it into the processor architecture.

Additional Coprocessors

Due to the focus on random access that exists in the JPEG2000 standard, it is possible to partition the processing into tasks that are well suited to be performed on coprocessors. The only coprocessor currently being offered is the VLx. If additional coprocessors were available, codeblocks or even individual coding passes within codeblocks could be assigned to the coprocessors.

6.4 Summary

In the introduction, the following question was asked:

> *Can the advantages of the JPEG2000 standard overcome the algorithm complexity to become widely adopted in high performance digital imaging applications?*

The work in this thesis used specialized DSP processors in a multiprocessor environment to design a high-performance JPEG2000 codec. Although this implementation is faster than existing software, it may not be fast enough at this point to spur a transition from existing formats such as JPEG.

Multiprocessor systems are very effective in accelerating the JPEG2000 format in print resolution images. Using image tiles as the grain size for each processor provides a partitioning that requires little overhead. The BSP-15 is currently not well suited to perform the codeblock processing, but with future versions can see a significant improvement.

Bibliography

[1] M. D. Adams. The JasPer Project Homepage. Available at http://www.ece.uvic.ca/ mdadams/jasper/, May 2002.

[2] Micheal Allen Barry Wilkinson. *Parallel Programming (PP): Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 1998.

[3] M. W. Marcellin; M. J. Gormish; A. Bilgin; M. P. Boliek. An Overview of JPEG-2000. In *Proc. of IEEE Data Compression Conference*, pages 523–541, 2000.

[4] W. Lee C. Basoglu and J. O'Donnell. The Equator MAP-CA^{TM} DSP: An End-to-End Broadband Signal Processor^{TM} VLIW. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 12, pages 646–659, August 2002.

[5] A.N. Skodras C.A. Christopoulos and T. Ebrahimi. The JPEG 2000 Still Image Compression Standard. In *IEEE Signal Processing Magazine*, pages 36–58, September 2001.

[6] Te-Hao Chang; Chung-Jr Lian; Hong-Hui Chen; Jing-Ying Chang; Liang-Gee Chen. Effective hardware-oriented technique for the rate control of JPEG2000 encoding. In *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, pages II–684– II–687 vol.2, May 2003.

[7] Xerox Corporation. Docuset3 Image Test Suite. Available at http://amberweb.wrc.xerox.com/View/Collection49835, May 2002.

[8] T. Ebrahimi D. Santa Cruz. A study of JPEG 2000 still image coding versus other standards. In *Proc. of EUSIPCO 2000*, September 2000.

[9] Jaswinder P. Singh David E. Culler. *Parallel Computer Architecture (PCA): A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1999.

[10] T. R. Reed E. B. Christopoulou, A. N. Skodras and C. A. Christopoulos. On the JPEG2000 Implementation on Different Computer Platforms. In *Proc. SPIE*, August 2000.

[11] Ericsson EPFL, Canon Research France. JPEG 2000 Implementation in Java. Available at http://jpeg2000.epfl.ch/, September 2001.

[12] Inc. Equator Technologies. BSP-15 Product Brief. Available at http://www.equator.com, June 2002.

[13] John Hennessy and David Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, May 2002.

[14] Yu-Sen Lin Chang-You Hsieh Chih-Hsieh Hsia Jen-Shiun Chiang, Chun-Hau Chang. High-speed EBCOT with dual context-modeling coding architecture for JPEG2000. In *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, volume 3, pages 865–868, May 2004.

[15] ISO/IEC JTC1/SC29/WG1. Information Technology - JPEG 2000 image coding system - Part 3: Motion JPEG2000, July 2002.

[16] ISO/IEC JTC1/SC29/WG1. Information Technology - JPEG 2000 image coding system - Part 4: Conformance Testing, July 2002.

[17] ISO/IEC JTC1/SC29/WG1 N2678. JPEG 2000 Part I: 020719 (Final Publication Draft), July 2002.

[18] M. Boliek; ISO/IEC JTC1/SC29/WG1 N360. New work item proposal: JPEG2000 image coding system, June 1996.

[19] ISO/IEC JTC1/SC29/WG1 N505. Call for contributions for JPEG 2000 (JTC 1.29.14, 15444): image coding system, March 1997.

[20] J. H. Kasner M. W. Marcellin P. J. Sementilli, A. Bilgin. Wavelet TCQ: submission to JPEG-2000. In *Proc. of SPIE, Appl. of Digital Image Proc.*, pages 2–12, July 1998.

[21] W.B. Pennebaker and J.L. Mitchell. *JPEG – Still image compression standard*. Van Nostrand Reinhold.

[22] J. Fangtham T. Sanguankotchakorn. A new approach to reduce encoding time in EBCOT algorithm for JPEG2000. In *TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region*, volume 4, pages 1338–1342, October 2003.

[23] D. Taubman. Software Architectures for JPEG2000. In *Proceedings of the IEEE International Conference for Digital Signal Processing*, pages 197–200, July 2002.

[24] D. Taubman. Kakadu Software. Available at http://www.kakadusoftware.com, June 2004.

[25] D. Taubman and M. Marcellin. *JPEG2000 Image compression fundamentals, standards and practice*. Kluwer Academic Publishers, 2001.

[26] Chung-Jr Lian Hong-Hui Chen Liang-Gee Chen Te-Hao Chang, Li-Lin Chen. Computation reduction technique for lossy JPEG2000 encoding through EBCOT Tier-2 feedback processing. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 3, pages 85–88, June 2002.

[27] T.; Izumi T.; Onoye-T.; Nakamura Y. Tsutsui, H.; Masuzaki. High speed JPEG2000 encoder by configurable processor. In *Circuits and Systems, 2002. APCCAS '02. 2002 Asia-Pacific Conference on*, volume 1, pages 45–50, October 2002.

[28] J.; Tangting Sun Wei Yu; Fritts. An efficient packetization algorithm for JPEG2000. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages 22–25, September 2002.

[29] B.; Bayoumi-M.A. Yijun Li; Aly, R.E.; Wilson. Analysis and enhancements for EBCOT in high-speed JPEG2000 architectures. In *Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on*, volume 2, pages II–207 – II–210, August 2002.

[30] Junko Yoshida. JPEG2000 Net imaging spec sparks design work. Available at http://www.eetimes.com/showArticle.jhtml?articleID=12806800, May 2000.

[31] Kun-Bin Lee Chein-Wei Jen Yun-Tai Hsiao, Hung-Der Lin. High-speed memory-saving architecture for the embedded block coding in JPEG2000. In *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, volume 5, pages 133–136, May 2002.

Appendix A

Detailed Results

Document	Pg	1 BSP-15	2 BSP-15	Speedup	4BSP-15	Speedup
ArtCatalog	1	9.878	4.992	1.979	2.553	3.869
ArtCatalog	2	8.095	4.099	1.975	2.115	3.827
ArtCatalog	3	9.178	4.654	1.972	2.388	3.843
ArtCatalog	4	8.940	4.533	1.972	2.325	3.845
ArtCatalog	5	8.862	4.495	1.972	2.312	3.834
ArtCatalog	6	10.452	5.274	1.982	2.703	3.867
ArtCatalog	7	10.031	5.071	1.978	2.624	3.824
ArtCatalog	8	8.933	4.526	1.974	2.328	3.837
Barnsley	1	9.092	4.596	1.978	2.402	3.785
Barnsley	2	10.034	5.074	1.978	2.606	3.850
Barnsley	3	10.286	5.192	1.981	2.693	3.819
Barnsley	4	9.982	5.034	1.983	2.579	3.870
Castle	1	10.662	5.379	1.982	2.786	3.827
Dac97	1	7.589	3.841	1.976	1.998	3.797
Dac97	2	7.698	3.910	1.969	2.026	3.800
Dac97	3	9.502	4.800	1.980	2.486	3.822
Dac97	4	2.308	1.205	1.915	0.672	3.435
Dac97	5	5.872	2.980	1.970	1.575	3.729
Dac97	6	10.877	5.492	1.980	2.829	3.845
Dac97	7	8.455	4.281	1.975	2.207	3.831
Dac97	8	10.436	5.283	1.975	2.745	3.802
Dac97	9	10.069	5.081	1.982	2.629	3.830
Dac97	10	8.932	4.524	1.975	2.324	3.844
Dac97	11	11.549	5.820	1.984	2.969	3.890
Dac97	12	6.123	3.120	1.963	1.629	3.760
Dac97	13	8.016	4.060	1.974	2.098	3.820
Dac97	14	7.932	4.014	1.976	2.078	3.817

Table A.1: BSP-15 Execution Time (sec) for Lossless Compression

Document	Pg	1 BSP-15	2 BSP-15	Speedup	4 BSP-15	Speedup
Dac97	15	6.492	3.295	1.970	1.711	3.794
Dac97	16	5.626	2.861	1.966	1.522	3.697
Dac97	17	6.462	3.292	1.963	1.709	3.781
Dac97	18	8.120	4.108	1.977	2.134	3.805
Dac97	19	8.606	4.350	1.979	2.258	3.811
Dac97	20	9.367	4.739	1.976	2.461	3.806
Dac97	21	5.715	2.903	1.969	1.531	3.734
GardenBook	14	9.528	4.825	1.975	2.483	3.837
GardenBook	15	12.073	6.097	1.980	3.103	3.891
GardenBook	16	10.974	5.538	1.982	2.840	3.864
GardenBook	17	11.146	5.625	1.982	2.881	3.869
GardenBook	18	11.599	5.847	1.984	2.994	3.873
GardenBook	19	11.077	5.598	1.979	2.875	3.853
GardenBook	20	10.996	5.548	1.982	2.852	3.856
GardenBook	21	12.144	6.131	1.981	3.140	3.868
GardenBook	22	11.289	5.704	1.979	2.934	3.848
GardenBook	23	2.020	1.095	1.846	0.603	3.350
PoolHall	1	11.917	6.044	1.972	3.066	3.887
QpsAll	1	5.868	2.987	1.965	1.598	3.673
QpsAll	2	9.216	4.663	1.977	2.402	3.837
QpsAll	3	8.872	4.493	1.974	2.327	3.812
QpsAll	4	7.147	3.627	1.971	1.881	3.799
QpsAll	5	6.617	3.365	1.966	1.746	3.790
QpsAll	6	6.237	3.174	1.965	1.658	3.763
QpsAll	7	6.304	3.212	1.963	1.666	3.784
Relay	1	8.763	4.434	1.976	2.289	3.828
TwoYears	1	8.892	4.518	1.968	2.314	3.843

Table A.2: BSP-15 Execution Time (sec) for Lossless Compression

Document	Page	1 BSP-15	2 BSP-15	Speedup	4 BSP-15	Speedup
ArtCatalog	1	10.703	5.433	1.970	2.825	3.789
ArtCatalog	2	7.405	3.755	1.972	1.977	3.746
ArtCatalog	3	8.987	4.557	1.972	2.343	3.835
ArtCatalog	4	8.688	4.421	1.965	2.285	3.802
ArtCatalog	5	8.445	4.306	1.961	2.224	3.797
ArtCatalog	6	11.467	5.811	1.974	3.032	3.782
ArtCatalog	7	10.208	5.174	1.973	2.668	3.826
ArtCatalog	8	8.615	4.380	1.967	2.259	3.813
Barnsley	1	10.409	5.279	1.972	2.770	3.757
Barnsley	2	10.786	5.457	1.977	2.919	3.695
Barnsley	3	11.197	5.724	1.956	2.988	3.747
Barnsley	4	10.601	5.366	1.976	2.793	3.796
Castle	1	11.457	5.796	1.977	3.020	3.793
Dac97	1	6.783	3.446	1.969	1.783	3.804
Dac97	2	6.922	3.509	1.973	1.828	3.788
Dac97	3	8.983	4.548	1.975	2.332	3.852
Dac97	4	1.324	0.770	1.719	0.555	2.384
Dac97	5	5.096	2.608	1.954	1.384	3.682
Dac97	6	11.769	5.951	1.978	3.060	3.846
Dac97	7	7.629	3.866	1.973	1.990	3.833
Dac97	8	10.998	5.580	1.971	2.908	3.782
Dac97	9	10.809	5.474	1.975	2.832	3.817
Dac97	10	8.388	4.245	1.976	2.217	3.784
Dac97	11	13.519	6.867	1.969	3.598	3.758
Dac97	12	5.513	2.821	1.954	1.472	3.746
Dac97	13	7.126	3.598	1.981	1.867	3.817
Dac97	14	7.310	3.729	1.960	1.907	3.834

Table A.3: BSP-15 Execution Time (sec) for Lossless Expansion

Document	Page	1 BSP-15	2 BSP-15	Speedup	4 BSP-15	Speedup
Dac97	15	6.829	3.515	1.943	1.857	3.678
Dac97	16	4.775	2.451	1.948	1.322	3.612
Dac97	17	5.594	2.855	1.960	1.482	3.774
Dac97	18	7.284	3.686	1.976	1.901	3.831
Dac97	19	8.489	4.310	1.970	2.245	3.782
Dac97	20	8.982	4.576	1.963	2.363	3.801
Dac97	21	4.939	2.508	1.969	1.363	3.624
GardenBook	14	10.715	5.447	1.967	2.859	3.749
GardenBook	15	14.294	7.243	1.974	3.773	3.788
GardenBook	16	12.661	6.405	1.977	3.330	3.802
GardenBook	17	12.917	6.541	1.975	3.392	3.808
GardenBook	18	13.432	6.789	1.979	3.506	3.831
GardenBook	19	12.871	6.529	1.972	3.378	3.811
GardenBook	20	12.836	6.494	1.977	3.382	3.796
GardenBook	21	13.873	7.002	1.981	3.631	3.821
GardenBook	22	12.990	6.587	1.972	3.422	3.797
GardenBook	23	1.013	0.737	1.374	0.544	1.862
PoolHall	1	14.561	7.414	1.964	3.900	3.734
QpsAll	1	4.938	2.536	1.947	1.350	3.658
QpsAll	2	9.745	4.949	1.969	2.567	3.797
QpsAll	3	9.363	4.761	1.967	2.477	3.780
QpsAll	4	6.628	3.380	1.961	1.787	3.710
QpsAll	5	7.425	3.825	1.941	2.054	3.616
QpsAll	6	6.659	3.458	1.925	1.856	3.588
QpsAll	7	6.638	3.434	1.933	1.853	3.582
Relay	1	9.553	4.848	1.970	2.564	3.726
TwoYears	1	8.805	4.476	1.967	2.313	3.808

Table A.4: BSP-15 Execution Time (sec) for Lossless Expansion

Document	Page	BSP-15	Kakadu	Speedup	JasPer	Speedup
ArtCatalog	1	2.553	7.58	2.969	67.05	26.263
ArtCatalog	2	2.115	5.01	2.369	53.18	25.142
ArtCatalog	3	2.388	6.10	2.554	60.11	25.172
ArtCatalog	4	2.325	5.92	2.546	58.46	25.141
ArtCatalog	5	2.312	5.88	2.544	58.14	25.152
ArtCatalog	6	2.703	8.25	3.052	70.57	26.110
ArtCatalog	7	2.624	6.78	2.584	64.45	24.566
ArtCatalog	8	2.328	5.94	2.551	58.51	25.131
Barnsley	1	2.402	7.72	3.214	64.78	26.969
Barnsley	2	2.606	6.69	2.567	63.82	24.486
Barnsley	3	2.693	6.84	2.540	64.85	24.080
Barnsley	4	2.579	6.39	2.478	62.18	24.108
Castle	1	2.786	7.16	2.570	67.38	24.187
Dac97	1	1.998	4.62	2.312	49.07	24.555
Dac97	2	2.026	4.79	2.365	50.18	24.772
Dac97	3	2.486	5.72	2.301	59.33	23.865
Dac97	4	0.672	2.25	3.349	25.82	38.434
Dac97	5	1.575	3.75	2.382	41.65	26.451
Dac97	6	2.829	7.68	2.715	69.72	24.649
Dac97	7	2.207	4.71	2.134	52.90	23.967
Dac97	8	2.745	7.17	2.612	66.86	24.354
Dac97	9	2.629	7.06	2.686	65.48	24.910
Dac97	10	2.324	5.30	2.281	56.30	24.228
Dac97	11	2.969	9.89	3.332	80.01	26.953
Dac97	12	1.629	4.30	2.640	43.43	26.667
Dac97	13	2.098	4.45	2.121	50.76	24.191
Dac97	14	2.078	4.94	2.377	51.28	24.678

Table A.5: Reference Software Execution Time (sec) for Lossless Compression

Document	Page	BSP-15	Kakadu	Speedup	JasPer	Speedup
Dac97	15	1.711	5.57	3.255	49.66	29.022
Dac97	16	1.522	3.63	2.385	40.33	26.502
Dac97	17	1.709	3.95	2.311	43.81	25.633
Dac97	18	2.134	4.74	2.221	51.80	24.274
Dac97	19	2.258	5.74	2.542	56.64	25.083
Dac97	20	2.461	5.71	2.320	59.03	23.984
Dac97	21	1.531	3.62	2.365	40.48	26.447
GardenBook	14	2.483	6.80	2.739	61.72	24.858
GardenBook	15	3.103	10.08	3.249	81.13	26.148
GardenBook	16	2.840	8.25	2.905	71.40	25.140
GardenBook	17	2.881	8.63	2.996	73.17	25.400
GardenBook	18	2.994	8.35	2.789	73.21	24.449
GardenBook	19	2.875	8.64	3.005	72.86	25.344
GardenBook	20	2.852	8.72	3.058	72.51	25.427
GardenBook	21	3.140	8.37	2.666	74.79	23.821
GardenBook	22	2.934	8.46	2.883	73.07	24.905
GardenBook	23	0.603	2.09	3.466	24.86	41.227
PoolHall	1	3.066	11.19	3.650	85.14	27.769
QpsAll	1	1.598	3.89	2.435	42.00	26.289
QpsAll	2	2.402	6.21	2.585	59.19	24.640
QpsAll	3	2.327	6.27	2.694	58.86	25.291
QpsAll	4	1.881	4.76	2.530	48.88	25.981
QpsAll	5	1.746	6.33	3.625	52.38	30.000
QpsAll	6	1.658	5.69	3.433	49.54	29.888
QpsAll	7	1.666	5.59	3.355	49.11	29.474
Relay	1	2.289	7.23	3.158	61.27	26.766
TwoYears	1	2.314	6.28	2.714	59.02	25.508

Table A.6: Reference Software Execution Time (sec) for Lossless Compression

Document	Page	4 BSP-15	Kakadu	Speedup	Jasper	Speedup
ArtCatalog	1	2.825	5.82	2.060	67.05	23.736
ArtCatalog	2	1.977	3.08	1.558	53.18	26.905
ArtCatalog	3	2.343	4.14	1.767	60.11	25.651
ArtCatalog	4	2.285	3.87	1.693	58.46	25.580
ArtCatalog	5	2.224	3.86	1.735	58.14	26.137
ArtCatalog	6	3.032	6.36	2.097	70.57	23.272
ArtCatalog	7	2.668	4.67	1.751	64.45	24.159
ArtCatalog	8	2.259	3.91	1.731	58.51	25.897
Barnsley	1	2.770	6.26	2.260	64.78	23.385
Barnsley	2	2.919	4.68	1.603	63.82	21.863
Barnsley	3	2.988	4.86	1.626	64.85	21.702
Barnsley	4	2.793	4.39	1.572	62.18	22.265
Castle	1	3.020	4.95	1.639	67.38	22.309
Dac97	1	1.783	2.46	1.380	49.07	27.524
Dac97	2	1.828	2.56	1.401	50.18	27.458
Dac97	3	2.332	3.23	1.385	59.33	25.439
Dac97	4	0.555	1.35	2.431	25.82	46.497
Dac97	5	1.384	2.07	1.496	41.65	30.094
Dac97	6	3.060	5.51	1.801	69.72	22.783
Dac97	7	1.990	2.40	1.206	52.90	26.578
Dac97	8	2.908	4.99	1.716	66.86	22.996
Dac97	9	2.832	4.98	1.758	65.48	23.121
Dac97	10	2.217	3.10	1.399	56.30	25.399
Dac97	11	3.598	8.20	2.279	80.01	22.239
Dac97	12	1.472	2.61	1.773	43.43	29.510
Dac97	13	1.867	2.33	1.248	50.76	27.187
Dac97	14	1.907	2.82	1.479	51.28	26.895

Table A.7: Reference Software Execution Time (sec) for Lossless Expansion

Document	Page	4 BSP-15	Kakadu	Speedup	Jasper	Speedup
Dac97	15	1.857	4.30	2.316	49.66	26.743
Dac97	16	1.322	2.12	1.604	40.33	30.507
Dac97	17	1.482	2.10	1.417	43.81	29.555
Dac97	18	1.901	2.57	1.352	51.80	27.246
Dac97	19	2.245	3.65	1.626	56.64	25.234
Dac97	20	2.363	3.43	1.452	59.03	24.983
Dac97	21	1.363	1.95	1.431	40.48	29.701
GardenBook	14	2.859	4.99	1.746	61.72	21.592
GardenBook	15	3.773	8.30	2.200	81.13	21.501
GardenBook	16	3.330	6.24	1.874	71.40	21.443
GardenBook	17	3.392	6.73	1.984	73.17	21.569
GardenBook	18	3.506	6.34	1.808	73.21	20.883
GardenBook	19	3.378	6.77	2.004	72.86	21.570
GardenBook	20	3.382	6.86	2.028	72.51	21.441
GardenBook	21	3.631	6.08	1.675	74.79	20.600
GardenBook	22	3.422	6.52	1.906	73.07	21.356
GardenBook	23	0.544	1.27	2.334	24.86	45.682
PoolHall	1	3.900	9.79	2.510	85.14	21.832
QpsAll	1	1.350	2.14	1.585	42.00	31.107
QpsAll	2	2.567	4.10	1.597	59.19	23.062
QpsAll	3	2.477	4.42	1.784	58.86	23.761
QpsAll	4	1.787	2.87	1.606	48.88	27.358
QpsAll	5	2.054	5.08	2.474	52.38	25.506
QpsAll	6	1.856	4.62	2.489	49.54	26.692
QpsAll	7	1.853	4.44	2.396	49.11	26.497
Relay	1	2.564	5.46	2.129	61.27	23.895
TwoYears	1	2.313	4.28	1.851	59.02	25.522

Table A.8: Reference Software Execution Time (sec) for Lossless Expansion

Document	Page	4 BSP-15 Compress	JJ2000 Compress	Speedup	4 BSP-15 Expand	JJ2000 Expand	Speedup
ArtCatalog	1	3.575	27	7.552	3.908	93	23.796
ArtCatalog	2	2.993	22	7.350	2.909	72	24.755
ArtCatalog	3	3.367	24	7.127	3.527	77	21.830
ArtCatalog	4	3.261	24	7.360	3.260	89	27.304
ArtCatalog	5	3.250	23	7.077	3.178	77	24.226
ArtCatalog	6	3.765	28	7.436	4.295	78	18.161
ArtCatalog	7	3.591	26	7.240	3.799	77	20.267
ArtCatalog	8	3.307	23	6.956	3.436	70	20.372
Barnsley	1	3.482	26	7.468	7.146	83	11.615
Barnsley	2	3.553	25	7.036	4.461	82	18.383
Barnsley	3	3.671	26	7.082	4.218	75	17.781
Barnsley	4	3.526	25	7.089	4.951	75	15.148
Castle	1	3.751	28	7.464	4.973	79	15.886
Dac97	1	2.873	21	7.310	4.308	68	15.783
Dac97	2	2.910	21	7.217	3.163	66	20.868
Dac97	3	3.381	24	7.098	3.256	75	23.033
Dac97	4	1.516	12	7.918	1.420	53	37.332
Dac97	5	2.485	18	7.243	2.342	64	27.325
Dac97	6	3.838	29	7.557	4.211	79	18.760
Dac97	7	3.061	22	7.186	4.089	75	18.340
Dac97	8	3.718	27	7.262	5.578	84	15.059
Dac97	9	3.584	27	7.533	4.992	81	16.228
Dac97	10	3.230	24	7.430	3.451	75	21.733
Dac97	11	4.123	32	7.761	5.179	83	16.027
Dac97	12	2.500	19	7.601	2.408	69	28.656
Dac97	13	2.959	21	7.096	2.745	66	24.043
Dac97	14	2.967	21	7.079	2.947	79	26.809

* All time information in this table represents
total execution time including file I/O overhead

Table A.9: JJ2000 Software End-to-End Time (sec) for Lossless JPEG2000

Document	Page	4 BSP-15 Compress	JJ2000 Compress	Speedup	4 BSP-15 Expand	JJ2000 Expand	Speedup
Dac97	15	2.693	21	7.797	2.906	73	25.124
Dac97	16	2.399	18	7.503	2.493	67	26.871
Dac97	17	2.579	19	7.367	2.459	68	27.657
Dac97	18	3.016	34	11.273	2.854	66	23.127
Dac97	19	3.176	35	11.021	3.275	81	24.730
Dac97	20	3.357	34	10.129	3.299	87	26.368
Dac97	21	2.406	21	8.729	2.297	84	36.569
GardenBook	14	3.458	34	9.833	4.168	76	18.235
GardenBook	15	4.262	38	8.916	5.220	88	16.859
GardenBook	16	3.909	30	7.674	4.518	90	19.919
GardenBook	17	3.934	30	7.626	4.635	87	18.772
GardenBook	18	4.012	41	10.219	4.595	96	20.894
GardenBook	19	3.937	41	10.415	4.558	90	19.747
GardenBook	20	3.906	33	8.448	4.767	84	17.622
GardenBook	21	4.136	34	8.221	5.757	84	14.592
GardenBook	22	3.985	32	8.031	4.664	84	18.010
GardenBook	23	1.472	16	10.871	1.419	64	45.118
PoolHall	1	4.305	36	8.362	5.340	107	20.039
QpsAll	1	2.445	22	8.998	2.308	66	28.592
QpsAll	2	3.348	30	8.960	3.549	75	21.134
QpsAll	3	3.296	28	8.496	4.938	78	15.797
QpsAll	4	2.784	25	8.981	2.897	72	24.852
QpsAll	5	2.793	26	9.308	3.164	69	21.809
QpsAll	6	2.676	26	9.718	3.026	73	24.126
QpsAll	7	2.646	26	9.827	2.903	68	23.426
Relay	1	3.317	31	9.347	3.716	79	21.258
TwoYears	1	3.249	27	8.309	3.406	78	22.900

* All time information in this table represents
total execution time including file I/O overhead

Table A.10: JJ2000 Software End-to-End Time (sec) for Lossless JPEG2000